SYSTEMS INITIATIVE

Functional Safety Language Standard

Version 1.0 Draft

July 2026

Abstract: The Accellera Functional Safety Language (FSL) defines a standardized, implementation-level language and data model for representing, exchanging, and integrating Functional Safety (FS) information across the electronic systems development lifecycle. FSL provides formal constructs for authoring FMEDAs, modeling failure modes and safety mechanisms, expressing failure mode effects, defining safety requirements, and mapping safety data to design representations at multiple abstraction levels. The language is derived directly from the conceptual FMEDA data model, ensuring structural consistency, predictable interpretation, and compatibility with automated tooling. By unifying the representation of FS data, FSL enables interoperability across tools and organizations, supports traceability throughout hierarchical supply-chain layers, and facilitates automation in safety-aware design, verification, and implementation flows. FSL is designed to complement existing Functional Safety standards such as ISO 26262 and IEC 61508 by providing a consistent mechanism to capture and exchange the data those standards require, without redefining their normative requirements. This document specifies the syntax, semantics, and conformance rules of the Functional Safety Language.

Keywords: Accellera, FMEDA, failure modes, functional safety, FSL, IEC 61508, interoperability, ISO 26262, safety mechanisms, safety metrics, traceability.

Notices

Accellera Systems Initiative (Accellera) Standards documents are developed within Accellera and the Technical Committee of Accellera. Accellera develops its standards through a consensus development process, approved by its members and board of directors, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are members of Accellera and serve without compensation. While Accellera administers the process and establishes rules to promote fairness in the consensus development process, Accellera does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an Accellera Standard is wholly voluntary. Accellera disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other Accellera Standard document.

Accellera does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or suitability for a specific purpose, or that the use of the material contained herein is free from patent infringement. Accellera Standards documents are supplied **"AS IS."**

The existence of an Accellera Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of an Accellera Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change due to developments in the state of the art and comments received from users of the standard. Every Accellera Standard is subjected to review periodically for revision and update. Users are cautioned to check to determine that they have the latest edition of any Accellera Standard.

In publishing and making this document available, Accellera is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is Accellera undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other Accellera Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of Accellera, Accellera will initiate action to prepare appropriate responses. Since Accellera Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, Accellera and the members of its Technical Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of Accellera Standards are welcome from any interested party, regardless of membership affiliation with Accellera. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Accellera Systems Initiative.
8698 Elk Grove Blvd Suite 1, #114
Elk Grove, CA 95624
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. Accellera shall not be responsible for identifying patents for which a license may be required by an Accellera standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Accellera is the sole entity that may authorize the use of Accellera-owned certification marks and/or trademarks to indicate compliance with the materials set forth herein.

Authorization to photocopy portions of any individual standard for internal or personal use must be granted by Accellera, provided that permission is obtained from and any required fee is paid to Accellera. To arrange for authorization please contact Lynn Garibaldi, Accellera Systems Initiative, 8698 Elk Grove Blvd Suite 1, #114, Elk Grove, CA 95624, phone (916) 670-1056, e-mail lynn@accellera.org. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained from Accellera.

Suggestions for improvements to the *name* Standard are welcome. They should be posted to the *name* Community Forum at:

https://forums.accellera.org/forum/62-functional-safety-language-fsl-10-public-review-feedback/

The current Working Group web page is:

https://www.accellera.org/activities/working-groups/functional-safety

Introduction

This introduction is informative and not part of the draft Accellera Functional Safety Langage Standard

The introduction page is optional for Accellera standards and considered informative. It may be added to the standard to explain the context and history of the standard. In case the standard is integral part of family of associated standards, the relation to the other standards should be explained.

The purpose of this standard is to define a unified, interoperable, and automation-friendly language for representing, exchanging, and integrating Functional Safety (FS) data across the electronic systems development lifecycle. The Functional Safety Language (FSL) enables consistent creation, manipulation, and transfer of FMEDA-related information across tools, teams, and supply-chain layers, supporting the development of safety-critical hardware and software in accordance with established safety standards such as ISO 26262 and IEC 61508.

Existing functional safety standards define *what* must be achieved to ensure the safety of electrical and electronic systems, but they intentionally do not prescribe *how* FS data should be represented, exchanged, or integrated across tools and organizations. As described in the Accellera FSL Group White Papers, this gap has resulted in fragmented, proprietary, and often manual approaches to FMEDA creation, safety mechanism modeling, failure mode representation, and safety data exchange. FSL is designed to fill this gap by providing a standardized, implementation-level language that is consistent with, and supports the application of, ISO 26262 and IEC 61508, without redefining or altering the requirements of those standards.

Historically, safety analyses—including FMEDA, FMEA, FTA, DFA, safety requirements, and safety manuals—have been authored and exchanged using heterogeneous formats, spreadsheets, and tool-specific databases. This fragmentation has hindered automation, interoperability, traceability, and the ability to retarget safety analyses across different standards or supply-chain layers. As modern systems grow in complexity and are developed by distributed teams across IP, component, module, and system levels, the need for a consistent representation of FS data has become critical.

The FSL standard provides a command-based formalism for authoring FMEDAs, mapping safety data to design representations, assigning safety mechanisms, defining failure mode effects, and specifying safety metrics. The language is designed to support both *authoring use cases*—where safety engineers create and refine FMEDA content—and *exchange use cases*, where completed or partially completed FMEDAs are shared “as-is” between organizations.

By standardizing the representation of FS data, FSL enables:

- **Interoperability**, allowing different EDA tools and safety analysis environments to exchange and interpret FS data consistently.
- **Traceability**, ensuring that safety information can be tracked across refinements, updates, and supply-chain transitions.
- **Automation**, enabling FS-aware design, verification, and implementation flows that integrate safety analysis with design metrics, technology information, and verification results.

The Accellera FSL is intended to serve as a foundational component of a broader FS-aware ecosystem, supporting the creation of robust, defensible safety cases and enabling scalable development of safety-critical electronic systems.

Participants

At the time this Accellera standard was developed, the *Functional Safety* Working Group had the following active participants:

Alessandra Nardi, Chair
Bala Chavali, Vice Chair
Teo Cupaiuolo, Secretary
Mouadh Ayache, Technical Editor
Louei Nefzi, Technical Editor

Alexandre Palus
Alexis Boutillier
Anamaria Hutuleac
Antonino Armato
Dheeraj Lokam
Ekaterina Vlasova
Evgeny Vlasov
Federico Venini
Francesco Sforza
Francesco Lertora
Franck Galtie
Ghani Kanawati
Giuseppe Capodanno
Ivano Shivananda Troja
Jacob Wiltgen
Jason Campbell
John Hayden
Jörg Grosse
Kaushik De
Kevin Rich
Louei Nefzi
Mark Hampton
Martin Barnasconi
Meirav Nitzan
Mouadh Ayache
Nikita Gulliya
Nir Maor
Om Ranjan
Pramod Bhardwaj
Regis Gubian
Riccardo Vincelli
Rolf Schlagenhaft
Shivakumar Chonnad
Shrenik Mehta
Stefano Lorenzini
Swami Venkat
Thiyagu Loganathan
Yakov Felikman
Yaswanth Kumar Kalla

The following members of the entity balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Agnisys, Inc.
ARM Ltd.
Arteris, Inc.
Cadence Design Systems, Inc.
Intel Corporation
NVIDIA Corporation
Siemens EDA
Synopsys

Contents

List of figures

List of tables

Functional Safety Language Standard

1. Overview

1.1 Scope

The Functional Safety Language (FSL) defines the syntax and semantics of a language and data model used to express Functional Safety (FS) intent in the development of safety-critical electronic systems, with an initial focus on FMEDA content. FSL supports the constructs and conventions to specify, validate, implement, and verify the safety-related data (e.g., failure modes, safety mechanisms, and safety metrics) and operations across the supply chain, from IP to system level.

FSL supports the application of existing FS standards such as ISO 26262 and IEC 61508 by providing a consistent way to capture and exchange the data those standards require, without redefining their requirements.

1.2 Purpose

The purpose of FSL is to enable automation, interoperability, and traceability in the creation and exchange of FS work products such as FMEDA, FTA, and DFA. It aims to unify the representation of FS data to facilitate integration across tools, teams, and lifecycle stages.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).[1, 2]

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

[1] The use of the word must is deprecated and cannot be used when stating mandatory requirements; must is used only to describe unavoidable situations.
[2] The use of will is deprecated and cannot be used when stating mandatory requirements; will is only used in statements of fact.

1.4 Key Characteristics

FSL provides a standardized, implementation-level method for representing and exchanging FS data. The following characteristics define the essential properties of FSL and guide its use across tools, organizations, and supply-chain layers.

1.4.1 Support for FMEDA creation and exchange

FSL provides constructs for describing:

- FMEDA structure
- failure modes
- safety mechanisms
- technology elements
- failure mode effects
- safety requirements
- mappings to design representations
- safety metrics and failure rates

The language supports both **authoring workflows** and **"as-is" exchange** of completed FMEDAs between suppliers and integrators.

1.4.2 Interoperability across tools and organizations

FSL enables consistent interpretation of FS data across:

- EDA tools
- safety analysis environments
- verification platforms
- suppliers and integrators

By standardizing the representation of FS information, FSL reduces tool-specific formats and eliminates ambiguity in data exchange.

1.4.3 Traceability across the safety lifecycle

FSL supports traceability of FS data across:

- refinements and updates
- hierarchical supply-chain layers (IP → component → module → system)
- safety work products (FMEDA, FMEA, FTA, DFA)

This traceability strengthens the integrity of safety cases and supports compliance with ISO 26262 and IEC 61508 traceability requirements.

1.4.4 Automation-friendly

FSL is designed to integrate with automated flows for:

- FMEDA generation
- design-to-safety mapping
- diagnostic coverage aggregation
- verification back-annotation
- safety metric computation

The language's structure enables FS-aware automation in design, verification, and implementation environments.

1.4.5 Data-model–driven

FSL is derived directly from the data model. Each construct in the language corresponds to an entity, attribute, or relationship in the data model, ensuring:

- structural consistency
- predictable interpretation
- compatibility with automated tooling

This data-model foundation enables reliable exchange and integration of FS information across the development lifecycle.

1.4.6 Scalable across abstraction levels

FSL supports safety analysis at multiple levels of abstraction:

- IP
- component
- module
- system

It enables hierarchical FMEDAs and consistent roll-up of safety data across the supply chain.

1.4.7 Technology-agnostic and representation-agnostic

FSL does not assume a specific design representation. It supports mapping to:

- functional models
- structural models
- RTL or gate-level netlists
- system-level models (e.g., SysML, IP-XACT)

This flexibility allows FSL to be used throughout the design and safety lifecycle.

1.4.8 Extensible

FSL supports user-defined arguments and collections, enabling organizations to extend the language to meet domain-specific or project-specific needs without compromising interoperability.

1.4.9 Alignment with existing Functional Safety standards

FSL is designed to support the implementation of established FS standards such as ISO 26262 and IEC 61508. These standards define *what* must be achieved for safety but do not specify *how* FS data should be represented, exchanged, or automated. FSL fills this gap by providing consistent language for capturing the data required by these standards, without redefining or altering their requirements.

1.5 Contents of This Standard

The organization of the remainder of this standard is as follows:

- Clause 2 identifies the normative references that are indispensable for the application of this Standard.
- Clause 3 defines the terms, definitions, acronyms, and abbreviations used throughout this Standard.
- Clause 4 introduces the FMEDA concepts and the structural data model underlying the FSL. It describes the FMEDA process, the design-representation context, and the conceptual data model from which the language is derived.
- Clause 5 specifies the FSL, including its language basics, data types, command structure, well-formedness rules, and conformance requirements.
- Annex A lists a bibliography of potentially useful additional reference material.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

[1] ISO 26262:2018, Road vehicles — Functional safety, Parts 1–12. International Organization for Standardization, Geneva, Switzerland, 2018.[3]

[2] IEC 61508:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems, Parts 1–7. International Electrotechnical Commission, Geneva, Switzerland, 2010.[4]

[3] IEC 62380:2004, Reliability data handbook — Universal model for reliability prediction of electronics components, PCBs and equipment. International Electrotechnical Commission, Geneva, Switzerland, 2004.[4]

[4] IEC 61709:2017, Electric components — Reliability — Reference conditions for failure rates and stress models for conversion. International Electrotechnical Commission, Geneva, Switzerland, 2017.[4]

[5] SN 29500 (series), Failure rates of components [Part1, Part2, Part 3]. Siemens AG, Munich, Germany.

[6] JEDEC Standard JESD89B, Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices. JEDEC Solid State Technology Association, Arlington, VA, USA, 2021.[5]

[3] Available at https://www.iso.org/standard/68383.html
[4] IEC publications are available from the International Electrotechnical Commission (http://www.iec.ch) and the American National Standards Institute (http://www.ansi.org/).
[5] JEDEC publications are available from the Global Standards for the Microelectronics Industry (http://www.jedec.org/).

3. Definitions, acronyms, and abbreviations

3.1 Introduction

For the purposes of this document, the following terms and definitions apply. The IEEE Standards Dictionary Online should be consulted for terms not defined in this clause.[6]

3.2 Definitions

3.2.1 Core Information Object Types

Entity: An Entity represents a primary information object in the data model. Entities correspond to distinct conceptual elements that exist independently and possess their own identity within an FMEDA or FS analysis context. Entities contain attributes and may participate in one or more Relationships.

Relationship: A Relationship represents an association between two or more information objects. Relationships capture how Entities are connected, constrained, or interact within the data model. A Relationship does not exist independently of the Entities it relates.

Weak Entity: A Weak Entity represents a dependent information object whose identity is defined by another Entity or Relationship. Weak Entities do not exist independently and cannot be created in isolation; instead, they are defined only in the context of another object.

3.2.2 Foundational Language Concepts

Object: An Object is a primary information object defined by the data model, corresponding to an Entity explicitly created through FSL create_* commands.

Argument: An Argument is a language construct (such as a command-line flag, switch, or option) passed to a command. Arguments supply the input values, object references, or configuration details necessary to execute the command and populate the attributes of the target data model object.

Command: A Command is a language construct used as an interface to create or add information within an instance of the data model. All commands follow specific naming and structural rules and may reference previously created objects. A command consists of a command name and zero or more arguments. The create_* commands also require an object name.

Instance: An Instance is a concrete realization of the data model created using the commands defined by this language. An instance contains Entities, Relationships, and Weak Entities arranged according to the rules of the model. Instances are evaluated for well-formedness and conformance to determine whether they satisfy the requirements of this standard.

Well-formed: An instance is well-formed when it satisfies all lexical, syntactic, and structural requirements defined by this standard. A well-formed instance uses valid command names, provides all required arguments with correct data types and in-range values, and respects all object reference and uniqueness constraints. Well-formedness is evaluated prior to semantic checking.

Ill-formed: An instance is ill-formed when it violates one or more lexical, syntactic, or structural requirements defined by this standard. This includes missing required arguments, using invalid command names, providing arguments of the wrong data type or out-of-range values, or violating object reference and uniqueness constraints. Ill-formed constructs are rejected prior to semantic or conformance checking.

[6] IEEE Standards Dictionary Online is available at: http://dictionary.ieee.org. An IEEE Account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.

Conforming: A conforming instance is a well-formed instance that additionally satisfies all semantic rules defined by this standard, including the command-specific conformance rules specified in 5.6. Conformance is a property of the instance itself, independent of any particular implementation that processes it.

Non-conforming: A non-conforming instance is one that fails to meet the requirements of this standard. All ill-formed instances are non-conforming, but an instance may also be non-conforming if it violates additional semantic constraints even when structurally complete.

Required Argument: A required argument is an argument that shall be provided whenever the corresponding command is invoked. Omission of a required argument renders the command syntax ill-formed and prevents the initialization of its corresponding data model attribute.

Optional Argument: An optional argument is an argument that may be omitted when invoking a command. If omitted, the default value defined by the command applies to the target attribute.

3.2.3 Further Concepts

Project: A Project is the top-level container for an FSL instance, encompassing one or more FMEDA objects and the associated shared objects (Technology Elements, Safety Mechanisms, Safety Requirements) used across those FMEDAs.

Library: A Library is a collection of reusable components that Projects may reference. Library components include Technology Elements, Safety Mechanisms, and Safety Requirements.

3.2.4 Structural and Language conventions

To ensure clarity and architectural precision, this standard maintains a strict separation between the command language syntax and the underlying data structures:

- Argument: This term is used exclusively within the context of the Language interface to describe the syntactic elements (flags, choices, and input variables) supplied during command execution.
- Attribute: This term is used exclusively within the context of the Data Model to describe the data fields, properties, and structural characteristics belonging to an instantiated object or entity.

A Command acts as the interface between these two domains; executing a language command processes its input arguments to populate, configure, or modify the respective attributes of objects within the data model.

3.3 Acronyms and abbreviations

The acronyms and abbreviations contained in this subclause are used throughout this LRM interface and within the underlying data model attributes.

ANSI American National Standards Institute

AoU Assumption of Use

ASIL Automotive Safety Integrity Level

BFR Base Failure Rate

CPU Central Processing Unit

DC Diagnostic Coverage

DFA Dependent Failure Analysis

DSL Domain Specific Language

DUA Design Under Analysis

EDA Electronic Design Automation

ER Entity-Relationship

FIT Failures in Time (failures per 10^9 hours)

FM Failure Mode

FME Failure Mode Effect

FMEA Failure Modes and Effects Analysis

FMEDA Failure Mode Effects and Diagnostic Analysis

FR Failure Rate

FS Functional Safety

FSL Functional Safety Language

FTA Fault Tree Analysis

IEC International Electrotechnical Commission

IEEE Institute of Electrical and Electronics Engineers

IP Intellectual Property

IP-XACT Intellectual Property eXchange-Advanced Configuration and Integration of IP components

ISO International Organization for Standardization

JEDEC Joint Electron Device Engineering Council

LFM Latent Fault Metric

LRM Language Reference Manual

MBSE Model-Based Systems Engineering

MPF Multiple Point Fault

NSR Not Safety-Related

OMG Object Management Group

PCB Printed Circuit Board

PMHF Probability Metric for random Hardware Failures

PVSG Potential to Violate a Safety Goal

RTL Register-Transfer Level

SEooC Safety Element out of Context

SET Single Event Transient

SFF Safe Failure Fraction

SG Safety Goal

SIL Safety Integrity Level

SM Safety Mechanism

SoC System on Chip

SPF Single Point Fault

SPFM Single Point Fault Metric

SPICE Simulation Program with Integrated Circuit Emphasis

SR Safety-Related

SREQ Safety Requirement

SV SystemVerilog

SysML Systems Modeling Language

TE Technology Element

Tcl Tool Command Language

TLSR Top-Level Safety Requirement

UPF Unified Power Format

VHDL VHSIC Hardware Description Language

4. FMEDA Process, Data Model, and Language

4.1 Introduction

This standard is designed to support distributed development across the supply chain, enabling system integrators, module suppliers, component vendors, and IP providers to perform safety activities at their respective hierarchy levels. To make this collaboration effective, this standard defines a structured way to capture, organize, and exchange safety information between stakeholders. This ensures that analyses such as FMEDA can be carried out independently while still integrating cleanly, supporting automation, interoperability, and traceability across diverse development environments.

This standard is designed to align with established functional-safety standards such as ISO 26262 and IEC 61508, and to facilitate their practical implementation. Accordingly, the definitions and calculations in this standard are intended to be consistent with those standards unless explicitly stated otherwise.

Safety activities span several forms of analysis and operations described in the Accellera Functional Safety White Paper (May 2021), including FTA, DFA, and FMEDA. The scope of this LRM is to define a standardized language for representing and exchanging FMEDA data.

4.2 FMEDA process

4.2.1 Overview

The FMEDA process is a bottom-up, inductive analysis that identifies how elements of a system can fail and how the effects of random hardware faults can be mitigated, either detected or controlled, to maintain or reach a safe state. It also produces safety metrics such as SPFM, LFM, and PMHF. This section describes the FMEDA process to identify the data and operations that guide the definition of the data model from which this standard is derived.

4.2.2 Flat FMEDA

The components and operations of an FMEDA are listed below and illustrated in Figure 1:

- **Input:** The primary input is a representation of the **Design Under Analysis (DUA)**, consisting of a comprehensive list of its components, typically organized hierarchically.
- **Intended functionality analysis:** This step identifies the elements of the **Functional Safety Analysis Hierarchy**, including the portions of the design that are safety-related and have the potential to violate a safety goal or fail to satisfy a safety requirement.
- **Failure Mode (FM) analysis:** For each safety-relevant element identified in the FS Analysis Hierarchy, all ways in which that element can deviate from its intended behavior are identified and classified as Failure Modes.
- **Technology Element (TE) identification:** For each failure mode of the DUA, one or more Technology Elements are selected from the **TE Library**. Each TE specifies the technology type and its associated Base Failure Rates (BFRs). This assignment may also be performed at the FS Analysis Hierarchy level.
- **Failure Mode Effects (FME) analysis:** This optional step identifies the impact of each failure mode of the DUA as observed when the DUA is integrated into the next level of the supply chain.
- **Safety Requirements (SREQ) mapping:** Failure modes may affect one or more Safety Requirements, defined by the application in which the DUA is deployed and collected in the **SREQ Library**. SREQs may be Safety Goals or Top-Level Safety Requirements. Safety metrics must be satisfied for each applicable Safety Requirement. This step is optional, for example when analyzing a Safety Element out of Context (SEooC).

— **Safety diagnostics allocation:** Safety mechanisms are selected from the **Safety Mechanisms (SM) Library** and applied to mitigate the identified failure modes, ensuring the system reaches or maintains a safe state within the required time.

— **Output:** The resulting safety metrics (Failure Rates, SPFM, LFM, PMHF) are calculated at multiple levels, such as per failure mode and for the overall FMEDA.

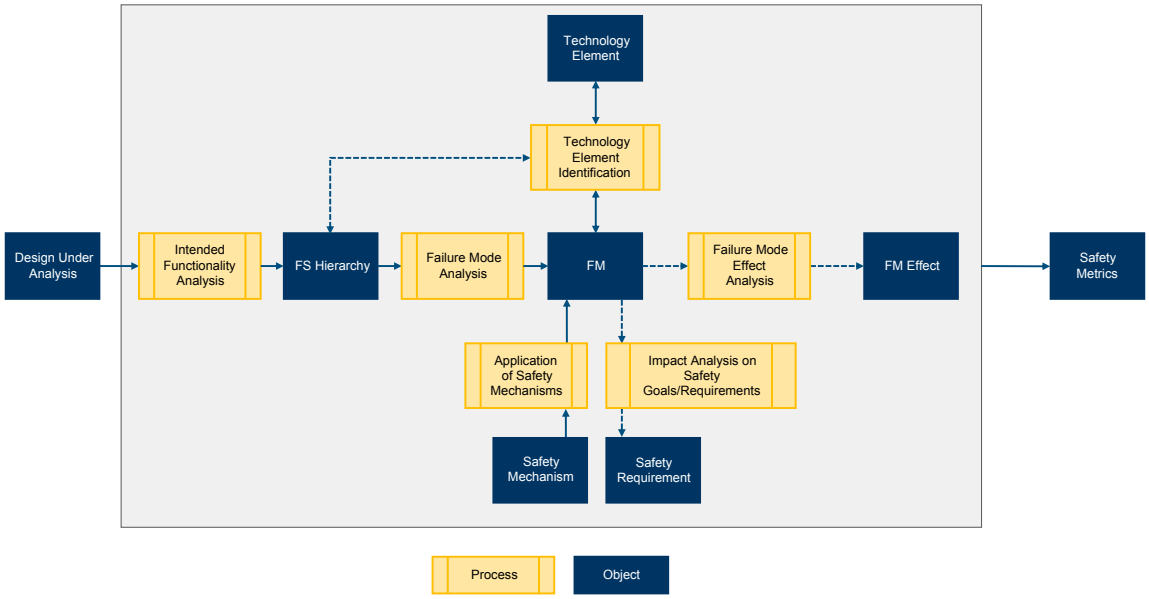

Figure 1—Fundamental data and operations in the creation of an FMEDA

The formalization of the FMEDA process shown in Figure 1 identifies the key elements involved and the relationships between them. These elements form the basis for defining the information categories of the FS data model and their interconnections.

Figure 2 expands the fundamental data and operations introduced in Figure 1 by showing how information is categorized and connected throughout the FMEDA process:

— **Input:** The DUA serves as the input to the FMEDA process and is external to the FS data model.

— **Processes:** Yellow rounded boxes represent the processes through which data is analyzed and generated.

— **Objects:** Blue rectangles represent objects in the FS data model. These objects correspond to information either generated during the FMEDA or provided as input. Some objects are defined within a specific FMEDA project, while others may be shared across multiple projects.

— **Relationships:** Grey parallelograms represent the connections or relationships between objects.

— **Outputs:** The outputs of the FMEDA are the safety metrics (Failure Rates, SPFM, LFM, PMHF), depicted with green rectangles. These metrics are calculated for various objects—such as the FMEDA, Elements, Failure Modes, and Failure Mode Effects—and may also be computed for each individual Safety Requirement.

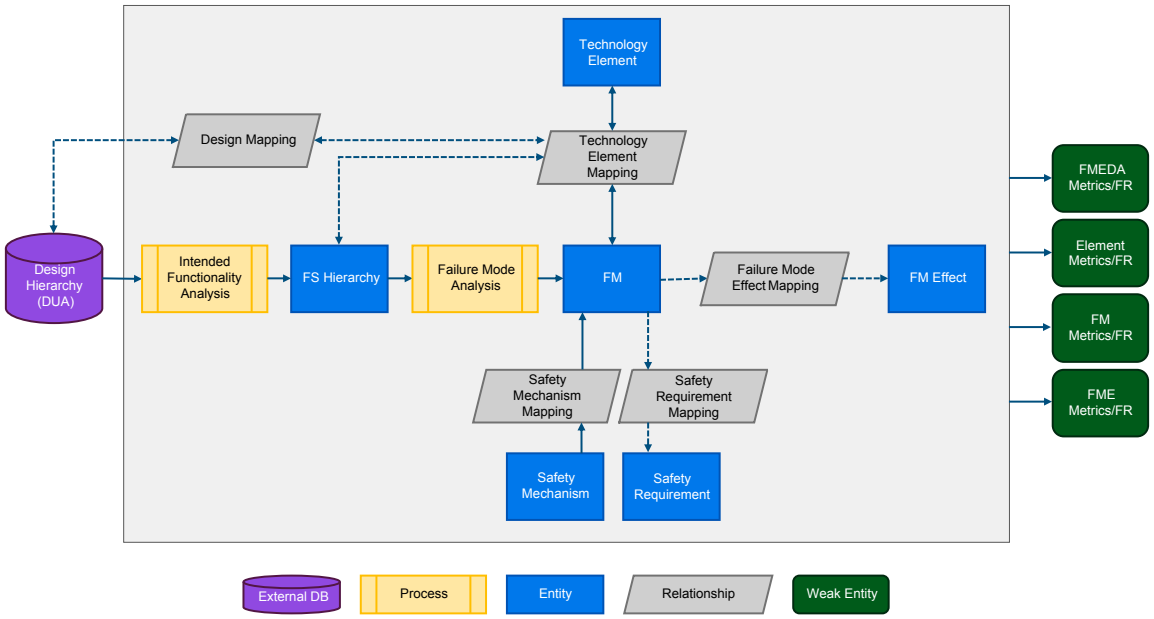

Figure 2—FMEDA process overview detailing data and processes

4.2.3 Hierarchical FMEDA

Hierarchical FMEDA provides a structured approach for combining FMEDA developed for different parts of a design, enabling scalable aggregation of analyses across multiple abstraction and integration levels.

The need to aggregate FMEDA results arises both within a single provider's workflow and across provider–integrator boundaries, where different teams may contribute portions of the overall safety analysis. As illustrated in Figure 3, the process begins from the DUA and can follow either a flat FMEDA approach—applying the full analysis flow directly to the complete design—or a hierarchical FMEDA approach, in which the DUA is partitioned into sub-blocks and each block is analyzed independently. These sub-block FMEDAs may be performed by different teams or organizations, enabling parallel development and supporting IP-level deliverables. The aggregated FMEDA then combines the results from the individual analyses, whether hierarchical or flat, to produce consistent system-level safety metrics aligned with the overall functional-safety objectives. Note that SMs, SREQs, and TEs are external to the FMEDA objects and may be shared across FMEDA projects; although they are depicted separately in Figure 3, they may refer to the same underlying objects.

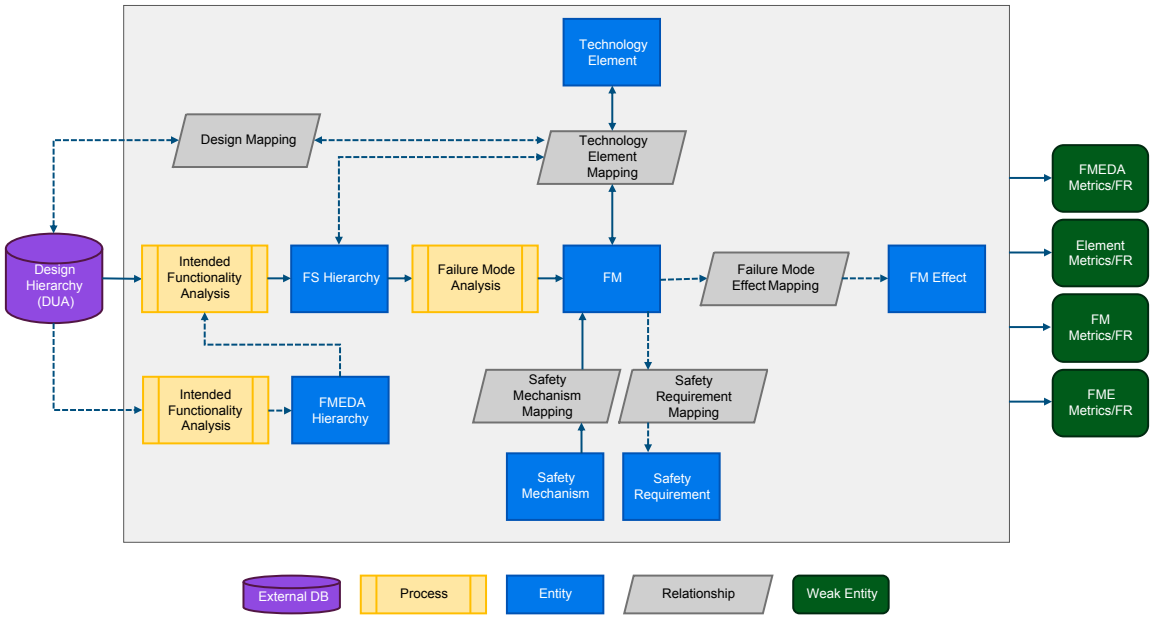

Figure 3—Hierarchical FMEDA process with mapping

4.2.4 Mapping of Data

In the context of data models and data bases, mapping is defined as the operation of connecting one set of data to another. In the context of the Data Model, mapping creates relationships and connects data across the FMEDA workflow defined in Figure 2 and Figure 3. Several types of mapping are defined in the context of performing an FMEDA, depending on the data it connects. The complete list is defined in Table 1.

Table 1—Data mapping involved in the FMEDA process

Type	Description
Technology Element Mapping	Maps the FS Hierarchy elements and/or the Failure Modes to the Technology Elements
Safety Mechanism Mapping	Links a Safety Mechanism to the Failure Mode it is meant to protect
Failure Mode Effects Mapping	Maps failure mode(s) to a failure mode effect that can be exported to an integrator for higher analysis levels
Design Mapping (FS Analysis Hierarchy)	Through the Technology Element Mapping, connects the design component(s) to the Functional Safety Hierarchy (represented as FMEDA Elements) identified during the analysis of the intended functionality
Design Mapping (Failure Modes)	Through the Technology Element Mapping, connects the design component(s) to the Failure Mode(s) identified during the FM analysis
Safety Requirements Mapping	Connects an FM to a Safety Requirement potentially violated by the FM itself

4.2.5 FMEDA Methodology Types: Assumption-Based and Calculation-Based

Two complementary methodologies are defined for determining the sizes of elements and failure modes, which are required to compute failure rates and safety metrics:

4.2.5.1 Assumption-Based FMEDA

In an assumption-based FMEDA, the sizes of elements and failure modes are specified directly by the safety analyst as numeric values, without reference to a design hierarchy mapping. The analyst provides the element size (via assign_te_element -element_size_*) and the failure mode size (via assign_te_fm with -fm_size_type_assumption_based set to absolute, percentage, or uniform_distribution). These user-supplied sizes take precedence when computing failure rates and safety metrics.

This methodology is appropriate when the design representation is not yet available, when working at an early lifecycle stage, or when performing a Safety Element out of Context (SEooC) analysis.

4.2.5.2 Calculation-Based FMEDA

In a calculation-based FMEDA, the sizes of elements and failure modes are derived automatically from the design hierarchy mapping provided via the -element_mapping and -fm_mapping arguments of assign_te_element and assign_te_fm, respectively. The mapping identifies the portion of the design netlist, RTL, or other structural representation that corresponds to the element or failure mode. Tool implementations shall extract the relevant area or bit-count metric from the mapped design representation to compute the element or failure mode size, with RTL-level mappings relying on estimation techniques as explicit physical metrics are not available.

For calculation-based FMEDA, -fm_size_type_calculation_based shall be set to one of the supported values (mapping_scaling, mapping_percentage, or mapping_uniform).

This methodology is appropriate when the design representation is available and the analyst can trace safety data to a formal design artifact.

Note–The -fmeda_type argument of define_fr_* and define_metric_* commands specifies which methodology (assumption-based or calculation-based) shall be used when computing the reported failure rate or metric. Both methodologies may coexist within a single project; the -fmeda_type selection at reporting time determines which size values are used.

4.3 Design Representation

4.3.1 Design Representation and FS language

The DUA is provided as a structured representation of the design, typically organized hierarchically. It may be expressed in any supported design-standard format, including SysML (OMG SysML v1.7 / v2.0), SystemC (IEEE 1666-2023), Verilog (IEEE 1364-2005), VHDL (IEEE 1076-2019), or the widely used Berkeley SPICE circuit-simulation format. Early in the development lifecycle the DUA may exist only as specification documents rather than a formal design representation.

The design representation is the primary input to the FS analysis process and remains independent of this standard. To support automation, this standard defines safety-specific content that complements—but does not replace or modify—the underlying design format. Maintaining this separation allows FS analysis to be performed independently of any particular representation and supports the diverse design practices found across the supply chain.

Connections between FS content and the design are established through mapping operations described in 4.2.4. These mappings allow analysts to associate safety mechanisms, failure modes, and other FS information with the appropriate elements in the design hierarchy.

4.3.2 Design Lifecycle and Representation Scope

As the design evolves, it may be represented in multiple forms—from high-level functional abstractions to detailed structural implementations—each suitable for safety analysis within a defined scope (e.g., IP, component, module, or system). FS analysis may be performed on any representation that fully covers the intended scope. Both functional and structural representations are supported, and the standard remains agnostic to the chosen DUA format.

This flexibility ensures traceability and consistency of safety information from system-level requirements down to IP-level implementations. FS analysts may group design elements into abstract or functional groups to aid in identifying failure modes and safety mechanisms, provided the representation is complete and no in-scope elements are omitted.

4.4 Data model for the language

The development of the functional-safety language defined in this standard follows a phased approach. First, the Functional Safety Analysis process is formalized, as shown in Figure 2 and Figure 3. Second, a conceptual data model is derived from the information exchanged and the operations defined in that process. Third, the Functional Safety language is formally derived from the conceptual data model.

4.4.1 Conceptual data model from the FMEDA process

Following the formalization of the FMEDA process described in clause 4.2, a conceptual data model is derived to represent the information required both to perform an FMEDA and to exchange an FMEDA report. The purpose of this model is to define and detail the functional-safety data needed to carry out the required activities and generate the associated work products, while deliberately avoiding any prescription of a reference implementation. It serves as a systematic foundation for defining a language.

The conceptual data model is captured using Entity–Relationship (ER) data modeling, a widely used method for representing the logical structure of information in a system in an abstract, implementation-independent way. In an ER model, the primary elements of the domain are captured as entities, their characteristics are expressed as attributes, and the associations between them are represented as relationships. In addition to regular entities, the model may include weak entities, which cannot be uniquely identified by their own attributes and therefore rely on the primary key of another entity for their identification. This modeling approach provides a clear and systematic representation of the information required by a process, supports consistent interpretation across tools and organizations, and forms a solid foundation for deriving a formal language or data-exchange format.

Figure 4 describes the high-level categories identified for the FMEDA process and the corresponding entities in the data model. The objects shown in Figure 4 use the same color coding as Figure 2 and Figure 3 to highlight the direct traceability between the FMEDA process and the conceptual data model required to implement that process. As a result, the data model is a traceable derivation of the data and data-mapping operations defined in the FMEDA process.

The DUA representation is not part of the data model, and the design mapping is intentionally excluded, as the design hierarchy itself is not part of the conceptual data model. Instead, the design-hierarchy information is captured within the TE Mapping or FM Mapping relationships as an attribute.

Each entity or relationship contains a set of attributes that define its properties.

Functional Safety Data Model				
FMEDA Process Data	Data Model Name	Information Type	Required	Functional Safety Language
FMEDA	FMEDA	Entity	Yes	`create_fmeda`
FS Hierarchy	Element	Entity	Yes	`create_element`
FM	FM	Entity	Yes	`create_fm`
Technology Element	TE	Entity	Yes	`create_te`
Safety Mechanism	SM	Entity	Yes	`create_sm`
FM Effect	FME	Entity	No	`create_fme`
Safety Requirement	SREQ	Entity	No	`create_sreq`
SM Mapping	SM_FM	Relationship	Yes	`assign_sm_fm`
FME Mapping	FM_FME	Relationship	No	`assign_fm_fme`
TE Mapping	TE_FM	Relationship	Yes	`assign_te_fm`
TE Mapping	TE_Element	Relationship	No	`assign_te_element`
Design Mapping	Inside TE_FM (*)	Relationship	No	`assign_te_fm -mapping -mapping_exclude`
Design Mapping	Inside TE_Element (*)	Relationship	No	`assign_te_element -mapping -mapping_exclude`
SREQ Mapping	FM_SREQ	Relationship	No	`assign_fm_sreq`
Calculated FR	FR_ISO26262	Weak Entity	No	`define_fr_iso26262`
Calculated Metrics	Metric_ISO26262	Weak Entity	No	`define_metric_iso26262`
Calculated FR	FR_IEC61508	Weak Entity	No	`define_fr_iec61508`
Calculated Metrics	Metric_IEC61508	Weak Entity	No	`define_metric_iec61508`

(*) (no Design Hierarchy in the data model)

Figure 4—Information in the data model derived from the FMEDA process

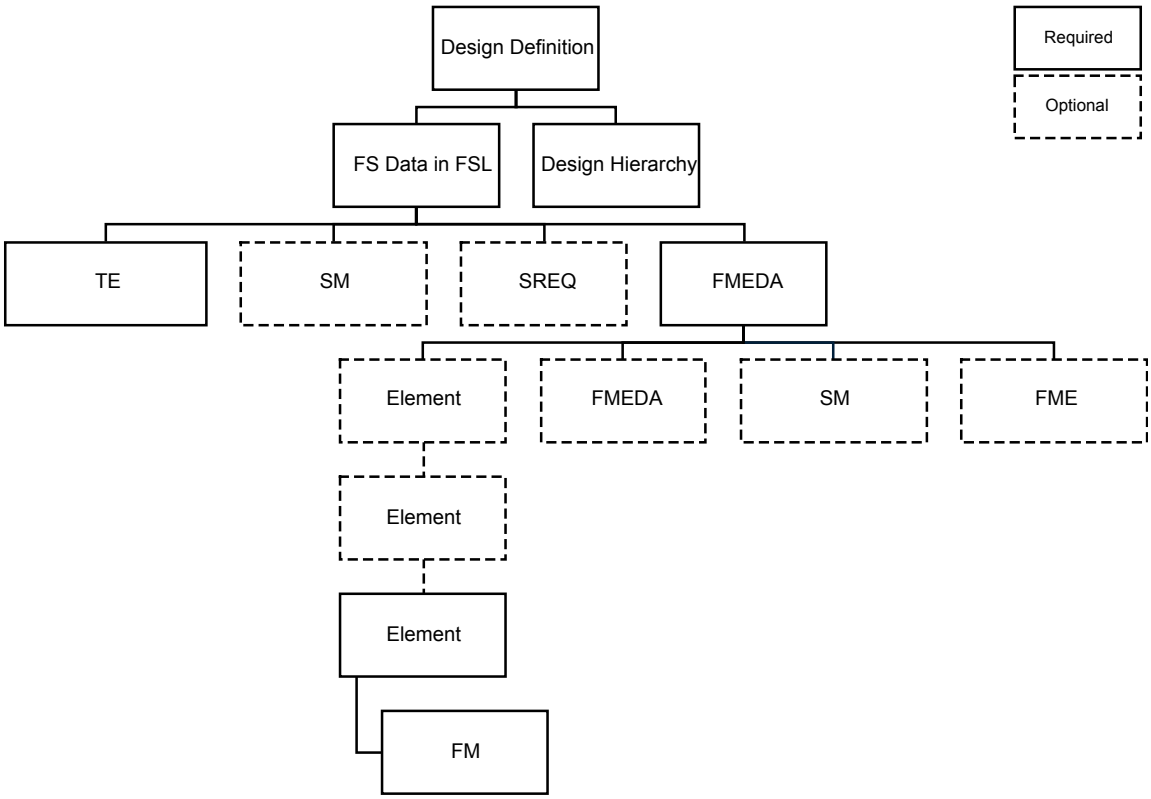

Figure 5—Design Definition and scope of the FSL data model objects

4.4.2 Data Model Scope, Required Entities and FSL Dependencies

As noted in Clause 4.2.5, this standard incorporates safety-specific information that complements the design

representation, ensuring that the complete set of data required for Functional Safety activities is available. Figure 5 illustrates the scope of the design hierarchy alongside the objects included in the ER-based conceptual data model, which serves as the foundation for deriving this standard.

Figure 4 identifies which entities are required, while Figure 6 captures the dependency rules that determine when one entity must exist prior to the creation of another. Within the data model, entities within an FMEDA scope form a hierarchy. FMEDAs and Elements may be nested creating a FMEDA hierarchy and an Element hierarchy respectively, and failure modes are always associated with a specific element.

The dependency order for FSL commands is as follows:

- create_fmeda establishes an FMEDA context. It may optionally reference a previously created FMEDA via -master_fmeda (instantiation) or -parent_fmeda (hierarchical FMEDA structure).
- create_element depends on an existing FMEDA. It may optionally reference an existing element via -parent_element to form an element hierarchy.
- create_fm depends on an existing element. The dependency is established through the -parent_element parameter.
- create_te, create_sreq, and create_sm introduce project-scope entities. They do not depend on any FMEDA and may appear anywhere before they are referenced.
- create_sm may optionally be associated with a specific FMEDA scope, but this is not required for creation.
- create_fme introduces a project-scope failure-mode effect. It depends only on the FMEDA project context being established and does not require a failure mode to exist at creation time.
- assign_te_element depends on a previously created Technology Element and Element.
- assign_te_fm depends on a previously created Technology Element and Failure Mode.
- assign_sm_fm depends on a previously created Safety Mechanism and Failure Mode.
- assign_fm_fme depends on a previously created Failure Mode and Failure Mode Effect.
- assign_fm_sreq depends on a previously created Failure Mode and Safety Requirement.
- define_fr_* and define_metric_* depend on all referenced entities and relationships being fully established prior to invocation.

	create_fmeda	create_element	create_fm	create_te	create_sm	create_fme	create_sreq	assign_fm_sreq	assign_sm_fm	assign_fm_fme	assign_te_element	assign_te_fm
create_fmeda	N/A											
create_element	√	N/A										
create_fm	√	√	N/A									
create_te				N/A								
create_sm					N/A							
create_fme	√					N/A						
create_sreq							N/A					
assign_fm_sreq	√	√	√				√	N/A				
assign_sm_fm	√	√	√		√				N/A			
assign_fm_fme	√	√	√			√				N/A		
assign_te_element	√	√		√							N/A	
assign_te_fm	√	√	√	√								N/A

Figure 6—Dependencies between entities in the ER data model, i.e. commands in FSL

4.5 Language derived from the data model

The language defined in this standard provides a set of constructs that serve as the abstract interface for constructing instances of the data model. The following naming rules apply to these constructs:

- Language constructs of the form create_* introduce information objects whose type is an **Entity**.
- Language constructs of the form assign_* introduce information objects whose type is a **Relationship**.
- Language constructs of the form define_* report values to information objects whose type is a **Weak Entity**.

These language constructs categories are also shown in the rightmost column of Figure 4.

The complete command reference, including syntax tables and conformance rules, is provided in Clause 5.

4.6 Principles for the development of the standard

The methodology used in this standard aligns closely with Model-Based Systems Engineering (MBSE) principles. It follows a model-driven progression in which the functional-safety analysis process is first formalized, the conceptual data model is then derived from that process, and the language is defined as a traceable realization of the underlying model. This reflects MBSE's emphasis on process-centric modeling, abstraction and separation of concerns, and the use of implementation-agnostic representations such as entities, attributes, and relationships. The approach also supports hierarchical decomposition and integration across abstraction levels, consistent with MBSE's multi-level system-architecture practices. By treating the conceptual data model as the authoritative source of truth and maintaining explicit traceability from process to model to language, the methodology is also aligned with Functional Safety standards best practices, which require clear, end-to-end traceability throughout the safety lifecycle.

5. Functional Safety Language (FSL)

5.1 Overview

FSL is a Domain Specific Language. It implements the syntax of the structural model defined in Clause 4 and the detailed properties of each information object. The definitions of the language well-formedness and conformance are introduced in 3.2.2. This clause defines the language basics and structure, the well-formedness rules (such as required attributes, and uniqueness) and the conformance rules (i.e. the language semantic). It also includes all the language constructs, attributes and partial examples of their usage.

5.2 Language Basics

5.2.1 Tcl Execution Model

FSL is defined as an extension of the Tool Command Language (Tcl). All FSL commands shall follow Tcl lexical and syntactic rules so that a standard Tcl interpreter can read and process FSL files.

The following requirements also apply:

- FSL commands shall be executed in the order in which they appear, consistent with Tcl command evaluation.
- Tcl features such as procedures, variables, and libraries may be used, provided that their behavior ultimately depends only on FSL commands for constructing or querying FSL data.
- FSL files shall not rely on proprietary or tool-specific Tcl extensions.
- After Tcl processing completes, the resulting state shall correspond to a well-defined sequence of FSL commands that construct the FSL instance.

5.2.2 Lexical Conventions

5.6 defines the syntax and semantics of all FSL commands. For each command, a syntax table defines the command's name and all of the command's valid arguments. The meta-syntax for the description of the syntax rules uses the conventions shown in Table 2.

Table 2—Lexical conventions

Visual cue	Represents
`courier`	The courier font indicates FSL code. For example, the following line: `create_fmeda FMEDA1`
bold	The **bold** font is used to indicate keywords that shall be typed exactly as they appear. For example, in the following command, the keyword **create_fm** shall be typed as it appears: **create_fm** *fm_name*
italic	The *italic* font represents user-defined FSL variables. For example, an element name shall be specified in the following line (after the **create_element** keyword): **create_element** *element_name*
list	*list* (or *xyz_list*) indicates a Tcl list, which is typically denoted with curly braces {….} or as a double-quoted string of elements "….". When a list contains only one non-list element (without special characters), the curly braces can be omitted, e.g., {a}, "a", and a are acceptable values for a single element. **-element_mapping** *element_mapping_list*

Visual cue	Represents
[ ] square brackets	Square brackets indicate optional arguments. For example, the following argument is optional: [**-user_defined_attribute** *element_list_of_tuples*]
< > angle brackets	Angle brackets (< >) indicate a grouping, usually of alternative arguments. For example, the following line shows the **estimated** or **measured** keywords are possible values for the **-dc_type** argument: **-dc_type** < **estimated** | **measured** >
| separator bar	The separator bar (|) character indicates alternative choices. For example, the following line shows the **sv**, **vhdl**, **spice** or **user-defined** keywords are possible values for the **-element_mapping_format** argument: **-element_mapping_format** <**sv** | **vhdl** | **spice** | **user-defined** >
boldface-red text	Syntactic keywords and tokens in the formal language definitions are shown in **boldface-red text**. For Example: **create_te** *te_name*

5.2.3 Data Types

The following data types are used in the command syntax tables defined in 5.6:

- **boolean**: A logical value specified as either TRUE or FALSE.
- **date**: A calendar date value adhering strictly to the ISO 8601 format (YYYY-MM-DD).
- **float**: An IEEE 754 double-precision floating-point number, represented by the Tcl double type.
- **hierarchical_path**: A text string representing a path through an object hierarchy, expressed as a dot-separated sequence of individual object names
- **integer**: A whole number used for counts, indexes, or discrete design metrics.
- **string**: A Tcl string value. Strings containing whitespace or special characters shall be enclosed in Tcl quoting braces ({ }) or double quotes (“”).
- **list_of_floats**: A Tcl list containing only float values.
- **list_of_tuples**: A Tcl list of name-value pairs, where each tuple is expressed as {name value}.
- **element_mapping_list**: A Tcl list of design hierarchy identifiers specifying components, nets, modules, or paths in the design representation. Entries shall conform to the naming rules of the format specified by -element_mapping_format.
- **element_mapping_exclude_list**: A Tcl list of design hierarchy identifiers specifying portions of the design to be excluded from mapping. Exclusions are applied after inclusions.

Numeric ranges denoted as [min, max] indicate inclusive lower and upper bounds. Ranges denoted as [0.0, n] indicate a non-negative float with no upper bound.

Note—Enumeration types are defined per-argument in the command syntax tables, and values shall match the specified literals exactly.

5.2.4 File Identification and Package Declaration

FSL files shall use the file extension .fsl. An FSL file is a valid Tcl script that shall be processable by a standard Tcl interpreter with the FSL package loaded.

An FSL file shall begin with the following package declaration as its first non-comment FSL statement:

package require fsl <version>

where <version> is a string corresponding to the revision of this standard being used. The package declaration serves as the file-level identifier and allows implementations to verify version compatibility before processing subsequent FSL commands.

If the package declaration is absent or specifies an unsupported version, the implementation shall issue a fatal error and terminate processing.

Note—The .fsl extension and package require convention follow the established practices of other Accellera domain-specific languages (DSLs), such as the Unified Power Format (UPF), enabling tool-agnostic recognition and processing of FSL files across electronic design automation (EDA) environments.

5.3 Language Structure, Naming Conventions, and Object References

5.3.1 Language Structure

Information in FSL shall be structured according to the data model defined in Clause 4. The hierarchy of objects is shown in Figure 5, and their dependencies are shown in Figure 6. An FSL instance shall consist of Entities, Relationships, and Weak Entities, each introduced through the commands defined in this standard.

5.3.2 Object creation

- Each object shall be created using a create_* command.
- The object_name shall be of type string and shall be unique within the scope of its parent object as defined by the data model hierarchy.
- An implementation shall report an error if a command attempts to create an object whose object_name conflicts with an existing object in the same scope.

Note—FSL is an append-only language. This standard defines commands for creating objects (create_*), establishing relationships (assign_*), and computing derived values (define_*). No commands are defined for deleting, modifying, or querying previously created objects. Once an object is created or a relationship is established, it is immutable for the remainder of the processing session. Implementations may provide tool-specific extensions for object inspection or modification, but such extensions are outside the scope of this standard and shall not affect the conformance of the FSL instance.

5.3.3 Object references

- Once created, an object shall be referenceable by subsequent commands.
- Object references shall be provided using the parameter corresponding to the object type (e.g., -fmeda_name, -fm_name, -sm_name, -te_name, -fme_name, -sreq_name) together with the parent path required by the data-model hierarchy, where applicable. Figure 7 identifies the valid scope and parent-reference chain for each entity object in the data model **(paths indicated in [] are optional).
- A reference shall resolve to exactly one previously created object within the applicable scope.
- An implementation shall report an error if a reference does not resolve uniquely.

	FMEDA Hierarchy	Element Hierarchy	Object Name
FMEDA	[parent_fmeda]		fmeda_name
Element	parent_fmeda	[parent_element]	element_name
FM	parent_fmeda	parent_element	fm_name
SM	[parent_fmeda]		sm_name
FME	parent_fmeda		fme_name
TE			te_name
SREQ			sreq_name

Figure 7—Scope in the data model hierarchy for each object

5.3.4 FMEDA chains and hierarchical FMEDA references

FMEDA objects may be organized in a hierarchical chain (e.g., System-FMEDA → Subsystem-FMEDA → Component-FMEDA).

The -parent_fmeda argument shall be used to reference an FMEDA within such a chain.

The value to -parent_fmeda shall be the full FMEDA chain, expressed as a dot-separated sequence of FMEDA names. Example: TOP_FMEDA.POWER_FMEDA.REGULATOR_FMEDA.

The dot-separated chain shall uniquely identify the referenced FMEDA within the project's FMEDA hierarchy.

An implementation shall report an error if the chain does not correspond to a valid sequence of ancestor FMEDAs.

The -master_fmeda argument instantiates a previously defined FMEDA. Instantiation does not establish a parent–child relationship. The hierarchical placement of the newly created FMEDA—identified by its fmeda_name—is determined exclusively by the -parent_fmeda argument, if provided.

5.3.5 Element chains and hierarchical element references

Elements may be created in a hierarchical chain (e.g., Part → Subpart → Subsubpart).

The -parent_element argument shall be used to reference an element within such a chain.

The value to -parent_element shall be the full element chain, expressed as a dot-separated sequence of element names. Example: TOP.MMU.TLB.

The dot-separated chain shall uniquely identify the referenced element within the FMEDA's element hierarchy.

An implementation shall report an error if the chain does not correspond to a valid sequence of ancestor elements.

5.3.6 Scope

A Project may contain one or more FMEDA objects. Technology Elements, Safety Mechanisms, and Safety Requirements are defined at the project level and may be shared across multiple FMEDA objects within the same project.

Note—The Project is an implicit scope defined by the set of FSL commands evaluated within a single Tcl processing context. No explicit create_project command is required. All objects created during a single evaluation session belong to the same project unless a conforming implementation provides explicit project-boundary mechanisms.

5.3.7 Error Handling

An implementation of this standard shall support error reporting when a violation of a mandatory rule occurs (indicated by "shall" language). An error shall prevent the erroneous construct from being incorporated into the data model. Whether processing continues after an error or halts immediately is implementation-defined, but the resulting instance shall not include the rejected construct.

5.4 Well-Formedness Rules

This clause defines the conditions under which a language construct or sequence of language constructs is considered well-formed. A well-formed language construct satisfies all lexical, syntactic, and structural requirements defined in this standard. A language construct that is ill-formed shall be rejected by an implementation of this standard prior to semantic or conformance checking.

A language construct is well-formed if and only if:

- Its name is valid and appears in the set of language constructs.
- All required arguments are present.
- Optional arguments may be omitted.
- No unknown or undeclared arguments are used.
- All provided arguments use the correct data type.
- All enumeration values match exactly the literals.
- All numeric values fall within the permitted ranges.
- No argument appears more than once in the same instance of the language construct unless explicitly permitted.

A language construct instance that violates any of the above conditions is ill-formed.

5.5 Conformance Rules

A command or sequence of commands is conforming if and only if it is:

- Well-formed according to 5.4, and
- Semantically valid according to the rules defined in 5.6.
- All objects referenced in the language construct instance have already been created.
- All objects referenced in the language construct instance satisfy scope and uniqueness rules.
- No command applied to the same object appears more than once.

A well-formed command that violates any semantic rule in 5.6 is non-conforming.

5.5.1 Conditionally Required Arguments

In the command syntax tables 5.6, certain arguments are listed as required but are conditional on the active safety standard context or specific object attributes. Implementations shall treat these arguments as conditionally required based on the following rules:

assign_fm_sreq

- Both -safety_relevant and -no_part are listed as required arguments.
- Rule_FMEDA_1 prohibits the use of -no_part in an ISO 26262 context.
- Rule_FMEDA_2 prohibits the use of -safety_relevant in an IEC 61508 context.
- The applicable argument is determined by the ASIL or SIL setting of the parent FMEDA.

assign_sm_fm

- The -active argument is listed as required.
- Rule_SM_FM_1 restricts its use to safety mechanisms where the target object's -configurable attribute is set to TRUE.
- When -configurable is FALSE, the -active argument shall not be specified.

create_sm

- The -dc_latent_primary and -dc_latent_secondary arguments are listed as required.
- Rule_SM_2 and Rule_SM_3 restrict them to ASIL-defined (ISO 26262) FMEDAs.
- For IEC 61508 FMEDAs, these arguments shall not be specified.

create_fm

- The -pvsg_permanent and -pvsg_transient arguments are specific to ISO 26262.
- The -no_effect_permanent and -no_effect_transient arguments are specific to IEC 61508.
- The applicable set of arguments is determined by the ASIL or SIL setting of the parent FMEDA.

assign_te_element

- The -element_size_* arguments and the -element_mapping argument are individually optional.
- However, at least one of these arguments is required for the command to be well-formed.

5.5.2 Mutually Exclusive Arguments

In the create_fmeda command, both -asil and -sil are present as required arguments.

- Exactly one of -asil or -sil shall be set to a value other than none, and the other shall be set to none.
- It shall be an error if both -asil and -sil are set to values other than none in the same command, as the ISO 26262 and IEC 61508 target attribute sets are mutually exclusive.
- It shall also be an error if both -asil and -sil are set to none, as every FMEDA shall be associated with at least one safety standard context.

5.5.3 Polarity Differences in Target Attributes

In the create_fm command, the language arguments -safety_relevant (ISO 26262) and -no_part (IEC 61508) pass values to target data model attributes that use opposite boolean polarity for the same conceptual classification:

- Setting -safety_relevant to TRUE indicates the failure mode is safety-relevant.
- Setting -no_part to TRUE indicates the failure mode is NOT part of the safety function (i.e., it is not safety-relevant).

The same structural polarity difference applies to the corresponding arguments in the assign_fm_sreq command. Implementers and users shall take care to interpret these arguments and their underlying data model attributes correctly in the context of the applicable safety standard.

5.6 FSL Commands

5.6.1 Introduction

This subclause specifies the complete set of FSL commands, organized by command category.

For each command, a syntax table defines the command name, purpose, arguments, types, and required/optional status. A companion conformance rules table defines the semantic rules that govern valid use of the command.

The commands are grouped as follows:

* Entity commands (create_*): create_fmeda, create_element, create_fm, create_sm, create_te, create_fme, create_sreq

* Relationship commands (assign_*): assign_sm_fm, assign_te_fm, assign_te_element, assign_fm_fme, assign_fm_sreq

* Weak Entity commands (define_*): define_fr_iso26262, define_fr_iec61508, define_metric_iso26262, define_metric_iec61508

Note 1—The define_* commands are reporting commands. They specify the parameters for failure rate and metric calculations and return computed values. The -fr_value and -metric_value attributes represent the output of these commands and are not user-supplied inputs.

Note 2—For create_* commands, the return value is an empty string on success, or a TCL_ERROR is raised on failure. For assign_* commands, the return value is an empty string on success, or a TCL_ERROR is raised on failure. For define_* commands, the return value is the computed -fr_value or -metric_value: failure rate values are expressed in FIT (Failures in Time, i.e., failures per 10^9 hours); SPFM and LFM metric values are expressed as percentages in [0.0, 100.0]; PMHF (ISO 26262) is expressed in FIT; and SFF (IEC 61508) is expressed as a percentage in [0.0, 100.0].

Note 3—The lexical conventions for command syntax table notation are defined in 5.2.2.

5.6.2 create_fmeda

Table 3—create_fmeda command

Purpose	Create a new FMEDA	
Syntax	**create_fmeda** *fmeda_name* [**-fmeda_description** *string*] **-asil** <**none** | **a** | **b** | **c** | **d** > **-sil** <**none** | **1** | **2** | **3** | **4** > [**-creator_name** *string*] [**-date** *date*] [**-version** *string*] [**-language_version** *string*] [**-master_fmeda** *fmeda_name*] [**-parent_fmeda** *parent_fmeda*] **-dc_aggregation_rule** <**max** | **sum** | **residual** > [**-comment** *string*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*fmeda_name*	The name of the FMEDA to be created
	-fmeda_description *string*	A textual description associated with the FMEDA
	-asil <**none** | **a** | **b** | **c** | **d** >	The ASIL target assigned to the FMEDA, defined according to ISO 26262

	-sil <**none** | **1** | **2** | **3** | **4** >	The SIL target assigned to the FMEDA, defined according to IEC 61508
	-creator_name *string*	The name of the organization responsible for generating the FMEDA
	-date *date*	The date of FMEDA creation
	-version *string*	The version identifier of the FMEDA object (e.g., 1.0, 2.3.1)
	-language_version *string*	The FSL version of the language used in the FMEDA definition
	-master_fmeda *fmeda_name*	The name of the FMEDA being instantiated by reference in the FMEDA hierarchy
	-parent_fmeda *parent_fmeda*	The identifier of the parent FMEDA to which this FMEDA is connected in the FMEDA hierarchy
	-dc_aggregation_rule <**max** | **sum** | **residual** >	The rule used to aggregate diagnostic coverage estimates across multiple safety mechanisms associated with the same failure mode. The aggregation rule applies across the entire project scope
	-comment *string*	Additional information associated with the FMEDA object
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_fmeda command defines a new *FMEDA* instance for safety analysis in accordance with ISO 26262 and IEC 61508.

- **-asil** and **-sil** — specify the applicable safety integrity level, thereby ensuring the *FMEDA* aligns with the requirements of the respective standard. Exactly one of **-asil** or **-sil** shall be set to a value other than none; setting both to non-none values is an error, as is setting both to none. The choice determines which standard-specific arguments and commands are applicable throughout the *FMEDA*.
- **-master_fmeda** — specifies the name of a master *FMEDA* being instantiated by reference in the *FMEDA* hierarchy, enabling modular composition and reuse of existing *FMEDA* definitions. When specified, this command creates an instance of the master *FMEDA* as a child within the *FMEDA* hierarchy. The *fmeda_name* (first positional argument) identifies this specific instance and must be unique within its parent context. The master *FMEDA* must have been previously defined.
- **-parent_fmeda** — specifies the identifier of the parent *FMEDA* to which this *FMEDA* is connected in the *FMEDA* hierarchy, establishing the hierarchical relationship between *FMEDAs*. This argument is only valid when **-master_fmeda** is also specified. When used together, **-master_fmeda** and **-parent_fmeda** create a hierarchical structure where *FMEDA* instances can be nested and organized independently of the element hierarchy.
- **-dc_aggregation_rule** — defines the heuristic used to combine diagnostic coverage estimates from multiple *safety mechanisms* applied to the same *failure mode*, supporting aggregation strategies such as max, sum, or residual.
- **-creator_name**, **-date**, **-version**, and **-language_version** — optional metadata that can be specified to enhance traceability, documentation, and project-specific customization.
- **-comment** — allows inclusion of additional information that does not fit into predefined fields.

A hierarchical *FMEDA* is created by specifying **-master_fmeda** to instantiate an existing *FMEDA* by reference, and **-parent_fmeda** to establish the hierarchical relationship. This enables modular composition and reuse of existing flat or hierarchical *FMEDA* elements. The hierarchical *FMEDA* architecture provides clean separation between element hierarchy and *FMEDA* hierarchy, allowing IP-level *FMEDAs* to be instantiated multiple times within subsystem or SoC-level *FMEDAs*. This pattern supports typical semiconductor design workflows where reusable IP blocks are integrated into larger systems while

maintaining consistent safety analysis across instances.

Rules

The following rules apply:

Table 4—Rules for 'create_fmeda' command

Rule ID	Rule
Rule_FMEDA_1	It shall be an error if both **-asil** and **-sil** are set to values other than none in the same **create_fmeda** command. The ISO 26262 and IEC 61508 argument sets are mutually exclusive; exactly one of **-asil** or **-sil** shall be set to a non-none value.
Rule_FMEDA_2	It shall be an error if both **-asil** is set to none and **-sil** is set to none. Every *FMEDA* shall be associated with at least one safety standard context.
Rule_FMEDA_3	When **-asil** is set to a, b, c, or d, and **-sil** is set to none, the following shall apply: - It shall be an error if the **create_element** command specifies **-element_type** with value component or subcomponent. - It shall be an error if the **create_fm** command uses the arguments **-no_effect_*** or **-no_part**. - It shall be an error if the **assign_fm_sreq** command uses the argument **-no_part**. - It shall be an error if any **define_*_iec61508** command is used.
Rule_FMEDA_4	When **-sil** is set to 1, 2, 3, or 4, and **-asil** is set to none, the following shall apply: - It shall be an error if the **create_element** command specifies **-element_type** with value part or subpart. - It shall be an error if the **create_fm** command uses any of the arguments **-pvsg_***, **-safety_relevant**, **-dc_latent_***, or **-perceived_***. - It shall be an error if the **assign_fm_sreq** command uses the argument **-safety_relevant**. - It shall be an error if any **define_*_iso26262** command is used.
Rule_FMEDA_5	The following apply based on **-dc_aggregation_rule**: - The aggregation shall apply only to the **-dc_*_estimated** arguments as defined by the **assign_sm_fm** command. - The **create_fm -dc_*_measured** arguments shall not be aggregated. Instead, aggregated measured diagnostic coverage (e.g., from fault injection activities) shall be directly defined via the **create_fm -dc_*_measured** arguments. - The **create_fm -dc_*_expert** arguments shall be used to directly define diagnostic coverage based on expert input, whether aggregated or not.
Rule_FMEDA_6	It shall be an error if the **-parent_fmeda** argument is used on an FMEDA that already contains child elements.
Rule_FMEDA_7	When an FMEDA is created with the **-master_fmeda** argument, modification of the instantiated FS Hierarchy is not permitted. In particular, its child objects shall not be created using the **create_element**, **create_fm**, **create_fme**, or **create_sm** commands. External objects (e.g., safety mechanisms, safety requirements, mappings, assignments, and metrics) may reference objects within the instantiated FMEDA. Such references do not constitute modification of the instantiated FMEDA.
Rule_FMEDA_8	When **-master_fmeda** is specified, it shall be an error if the safety standard of the instantiated FMEDA is inconsistent with that of the master FMEDA. If the master *FMEDA* is ISO 26262-based (i.e., **-asil** is set to a non-none value), then the instance shall also specify a non-none **-asil** value and its **-sil** shall be none; if the master *FMEDA* is IEC 61508-based (i.e., **-sil** is set to a non-none value), then the instance shall also specify a non-none **-sil** value and its **-asil** shall be none. Mixing ISO 26262 ASIL-based and IEC 61508 SIL-based *FMEDAs* within the same hierarchical *FMEDA* is not permitted.

How-to Examples

Example 1: Basic Flat FMEDA for ISO 26262 ASIL B

This example creates a basic flat FMEDA for an automotive SoC targeting ASIL B compliance with ISO

26262, using the max rule for diagnostic coverage aggregation.

```
create_fmeda CPU_Core_FMEDA \
        -fmeda_description {Flat FMEDA for CPU core safety analysis} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2025-12-10 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule max \
        -comment {Basic flat FMEDA for automotive CPU core} \
        -user_defined_arguments {}
```

Example 2: Flat FMEDA for IEC 61508 SIL 3

This example creates a flat FMEDA for an industrial controller targeting SIL 3 compliance with IEC 61508, using the sum rule for diagnostic coverage aggregation.

```
create_fmeda Industrial_Controller_FMEDA \
        -fmeda_description {Flat FMEDA for industrial safety controller} \
        -asil none \
        -sil 3 \
        -creator_name {Accellera FS WG} \
        -date 2025-12-10 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule sum \
        -comment {Industrial controller FMEDA with SIL 3 requirements} \
        -user_defined_arguments {}
```

Example 3: Hierarchical FMEDA Instantiation for ISO 26262 ASIL D

This example demonstrates hierarchical FMEDA instantiation using -master_fmeda and -parent_fmeda. First, prerequisite master FMEDAs are created, then an instance is created within a parent FMEDA hierarchy with all arguments specified.

```
# Prerequisite: Create master FMEDA (fully specified with all metadata)
create_fmeda CPU_Core_FMEDA \
        -fmeda_description {Master FMEDA for CPU core} \
        -asil d \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule residual \
        -comment {Master FMEDA for ASIL D CPU core with residual aggregation} \
        -user_defined_arguments {}

# Prerequisite: Create parent FMEDA
create_fmeda System_Level_FMEDA \
        -fmeda_description {System-level parent FMEDA} \
```

```
        -asil d \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule residual \
        -comment {Parent FMEDA for hierarchical MCU composition} \
        -user_defined_arguments {}

# Instantiate master FMEDA as child (all arguments specified)
create_fmeda MCU_Hierarchical_FMEDA \
        -fmeda_description {MCU instance derived from CPU_Core_FMEDA master} \
        -asil d \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda CPU_Core_FMEDA \
        -parent_fmeda System_Level_FMEDA \
        -dc_aggregation_rule residual \
        -comment {Hierarchical MCU instance for system-level integration} \
        -user_defined_arguments {}
```

Example 4: IP-Level FMEDA Creation and Instantiation for Subsystem Hierarchy

This example demonstrates creating an IP-level FMEDA and then instantiating it multiple times within a subsystem FMEDA hierarchy. This pattern enables modular IP reuse across different instances in complex semiconductor designs.

```
# Step 1: Create IP-level FMEDA for RAM
create_fmeda IP1_FMEDA \
        -fmeda_description {Reusable RAM IP FMEDA} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule max \
        -comment {IP-level FMEDA for RAM with Part1.SubPart1 hierarchy} \
        -user_defined_arguments {}

# Step 2: Create subsystem-level FMEDA
create_fmeda SUBSYSTEM_FMEDA \
        -fmeda_description {Subsystem integrating multiple RAM IPs} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
```

```
        -dc_aggregation_rule max \
        -comment {Subsystem FMEDA with hierarchical IP instantiation} \
        -user_defined_arguments {}

# Step 3: Instantiate IP1_FMEDA twice as child FMEDAs in subsystem
create_fmeda RAM1 \
        -fmeda_description {First RAM instance in subsystem} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda IP1_FMEDA \
        -parent_fmeda SUBSYSTEM_FMEDA \
        -dc_aggregation_rule max \
        -comment {RAM1 instance derived from IP1_FMEDA master} \
        -user_defined_arguments {}

create_fmeda RAM2 \
        -fmeda_description {Second RAM instance in subsystem} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-05-26 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda IP1_FMEDA \
        -parent_fmeda SUBSYSTEM_FMEDA \
        -dc_aggregation_rule max \
        -comment {RAM2 instance derived from IP1_FMEDA master} \
        -user_defined_arguments {}
```

Example 5: Hierarchical Instantiation with Different ASIL Levels

This example demonstrates instantiating a master FMEDA at a different ASIL level. The master FMEDA targets ASIL B, while the instance targets ASIL D—both adhering to ISO 26262 per Rule_FMEDA_8. This pattern enables IP reuse at varying safety integrity levels within the same standard.

```
# Prerequisite: Create master FMEDA with ASIL B
create_fmeda IO_Controller_FMEDA \
        -fmeda_description {Master IO controller FMEDA for ASIL B} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule max \
        -comment {Reusable IO controller master} \
        -user_defined_arguments {}

# Prerequisite: Create parent FMEDA for ASIL D system
```

```
create_fmeda Safety_Critical_System \
        -fmeda_description {Safety-critical system parent for ASIL D} \
        -asil d \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule residual \
        -comment {High-level ASIL D parent FMEDA} \
        -user_defined_arguments {}

# Instantiate with ASIL D (higher level than master ASIL B)
create_fmeda IO_Instance_ASIL_D \
        -fmeda_description {ASIL D instance of IO controller master} \
        -asil d \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda IO_Controller_FMEDA \
        -parent_fmeda Safety_Critical_System \
        -dc_aggregation_rule residual \
        -comment {IO instance upgraded to ASIL D within safety-critical system} \
        -user_defined_arguments {}
```

Example 6: Multi-Layer FMEDA Hierarchy

This example demonstrates a three-level hierarchical FMEDA structure. A memory IP master is instantiated within a cache subsystem master, which itself is then instantiated as a child of a top-level SoC parent. This nesting pattern enables modular composition of reusable components across system hierarchy levels.

```
# Layer 1: Create reusable memory IP master
create_fmeda Memory_IP_FMEDA \
        -fmeda_description {Reusable memory IP FMEDA} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule sum \
        -comment {Memory IP master} \
        -user_defined_arguments {}

# Layer 2: Create reusable cache subsystem master
create_fmeda Cache_Subsystem_Master \
        -fmeda_description {Cache subsystem FMEDA master} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
```

```
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule sum \
        -comment {Reusable cache subsystem master} \
        -user_defined_arguments {}

# Layer 3: Instantiate memory IP within cache subsystem
create_fmeda L2_Cache_Memory \
        -fmeda_description {L2 cache memory instance} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda Memory_IP_FMEDA \
        -parent_fmeda Cache_Subsystem_Master \
        -dc_aggregation_rule sum \
        -comment {Memory instance within cache subsystem} \
        -user_defined_arguments {}

# Layer 4: Create SoC-level parent
create_fmeda SoC_FMEDA \
        -fmeda_description {SoC-level parent FMEDA} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -dc_aggregation_rule sum \
        -comment {Top-level SoC system} \
        -user_defined_arguments {}

# Layer 5: Instantiate cache subsystem within SoC
create_fmeda L2_Cache_Subsystem_Instance \
        -fmeda_description {L2 cache subsystem instance in SoC} \
        -asil b \
        -sil none \
        -creator_name {Accellera FS WG} \
        -date 2026-06-03 \
        -version "1.0" \
        -language_version 1.0 \
        -master_fmeda Cache_Subsystem_Master \
        -parent_fmeda SoC_FMEDA \
        -dc_aggregation_rule sum \
        -comment {Cache subsystem instance within SoC} \
        -user_defined_arguments {}
```

5.6.3 create_te

Table 5—create_te command

Purpose	Create a technology element	
Syntax	**create_te** *te_name* [**-te_description** *string*] [**-te_type** <**digital** | **ram** | **rom** | **flash** | **analog** >] [**-source** <**iec_62380** | **sn_29500** | **iec_61709** | **jesd89** | **expert** >] **-fr_permanent** *float* [*0.0*, *n*] **-fr_transient** *float* [*0.0*, *n*] **-fr_transient_set** *float* [*0.0*, *n*] **-unit_design_element_size_permanent** *float*]*0.0*, *n*] **-unit_design_element_size_transient** *float*]*0.0*, *n*] **-unit_design_element_size_transient_set** *float*]*0.0*, *n*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*te_name*	The name of the technology element to be created
	-te_description *string*	A textual description associated with the technology element
	-te_type <**digital** | **ram** | **rom** | **flash** | **analog** >	The type of technology represented by the technology element
	-source <**iec_62380** | **sn_29500** | **iec_61709** | **jesd89** | **expert** >	The source of failure rate data used for the technology element
	-fr_permanent *float* [*0.0*, *n*]	The base failure rate for permanent faults associated with the technology element
	-fr_transient *float* [*0.0*, *n*]	The base failure rate for transient faults associated with the technology element
	-fr_transient_set *float* [*0.0*, *n*]	The base failure rate for single event transient faults associated with the technology element
	-unit_design_element_size_permanent *float*]*0.0*, *n*]	The area of the unit design element for permanent faults, used to calculate the raw failure in time value for elements or failure modes
	-unit_design_element_size_transient *float*]*0.0*, *n*]	The area of the unit design element for transient faults, used to calculate the raw failure in time value for elements or failure modes
	-unit_design_element_size_transient_set *float*]*0.0*, *n*]	The area of the unit design element for single event transient faults, used to calculate the raw failure in time value for elements or failure modes
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_te command defines a new *technology element* instance for use in FMEDA analysis.

- *te_name* — specifies the unique name assigned to the *technology element*.
- **-te_description** — provides an optional textual description to enhance documentation and traceability of the *technology element*.
- **-te_type** — defines the technology category, supporting digital, ram, rom, flash, and analog types to ensure comprehensive coverage of semiconductor elements.
- **-source** — indicates the origin of failure rate data, which may be derived from reference standards such as IEC 62380, SN 29500, IEC 61709, JESD89, or from expert input, thereby ensuring traceability and alignment with industry practices.

— **-fr_permanent**, **-fr_transient**, and **-fr_transient_set** — specify the base failure rates, expressed in FIT (failures in time), for permanent, transient, and single event transient faults, respectively. This enables detailed and differentiated fault modeling for the *technology element*.

— **-unit_design_element_size_permanent**, **-unit_design_element_size_transient**, and **-unit_design_element_size_transient_set** — specify the area of the unit design element of the *technology element* for permanent, transient, and single event transient fault types, respectively. These values are used to calculate the number of unit design elements per *element* or *failure mode*, which is essential for determining the raw failure in time (FIT) for each *element* or *failure mode*.

The calculation follows the unified formula, where * denotes the fault type (permanent, transient, or transient_set):

$$\#\text{unit\_design\_elements}_{*} = \frac{\text{FM\_size}_{*}}{\text{unit\_design\_element\_size}_{*}} \quad (1)$$

This unified formula applies to all *technology elements*, including digital, memory (ram, rom, flash), analog, and package or pin components. The formula accommodates different base failure rate units for permanent and transient faults, such as FR per area, FR per transistor, FR per pin, etc. By allowing a distinct value for the SET base failure rate, independent accounting for the single event transient (SET) contribution is supported.

Rules

The following rules apply:

Table 6—Rules for ‘create_te’ command

Rule ID	Rule
Rule_TE_1	When the technology element’s **-te_type** is ram, rom, or flash, the following constraints apply: -In **assign_te_fm**, the values of **-fm_size_permanent_*_based** and **-fm_size_transient_*_based** for the same failure mode shall be equal. -In **create_te**, the value of **-fr_transient_set** shall be 0. -In **assign_te_fm**, the value of **-fm_size_transient_set_*_based** shall be 0.
Rule_TE_2	When the technology element’s **-te_type** is set to analog, and assuming no contribution from transient faults, the following constraints apply: -In **assign_te_fm**, the value of **-fm_size_transient_*_based** shall be 0. -In **create_te**, the value of **-fr_transient_set** shall be 0. -In **assign_te_fm**, the value of **-fm_size_transient_set_*_based** shall be 0.
Rule_TE_3	The following constraints apply to all *technology elements*: - The values of **-unit_design_element_size_permanent**, **-unit_design_element_size_transient**, and **-unit_design_element_size_transient_set** shall be set to 1 if the *failure mode* size or *element* size is already expressed in number of unit design elements.

How-to Examples

Example 1: Digital Technology Element with IEC 62380 Failure Rates

This example creates a digital technology element using IEC 62380 standard failure rate data for a 28nm CMOS logic process, including permanent, transient, and SET fault contributions.

```
create_te Logic_28nm_Digital \
    -te_description {28nm CMOS digital logic technology element with IEC 62380 failure rates} \
    -te_type digital \
```

```
    -source iec_62380 \
    -fr_permanent 5.0 \
    -fr_transient 2.0 \
    -fr_transient_set 0.5 \
    -unit_design_element_size_permanent 1000.0 \
    -unit_design_element_size_transient 1000.0 \
    -unit_design_element_size_transient_set 1000.0 \
    -user_defined_arguments {}
```

Example 2: SRAM Technology Element with JESD89 Data

This example creates an SRAM technology element following Rule_TE_1 constraints, with matching permanent/transient sizes and zero SET contribution for memory technology.

```
create_te SRAM_1Mbit \
    -te_description {1Mbit SRAM with JESD89 failure rate database} \
    -te_type ram \
    -source jesd89 \
    -fr_permanent 10.0 \
    -fr_transient 8.0 \
    -fr_transient_set 0.0 \
    -unit_design_element_size_permanent 1.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0 \
    -user_defined_arguments {}
```

Example 3: Analog Technology Element with Expert Input

This example creates an analog technology element with expert-derived failure rates, following Rule_TE_2 with zero transient and SET contributions typical for analog circuits.

```
create_te Analog_OpAmp \
    -te_description {Operational amplifier analog technology with expert failure rate estimates} \
    -te_type analog \
    -source expert \
    -fr_permanent 15.0 \
    -fr_transient 0.0 \
    -fr_transient_set 0.0 \
    -unit_design_element_size_permanent 500.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0 \
    -user_defined_arguments {}
```

Example 4: Flash Memory Technology Element with SN 29500

This example creates a flash memory technology element using SN 29500 reliability standard, with Rule_TE_1 constraints for memory technology.

```
create_te Flash_64KB \
    -te_description {64KB Flash memory with SN 29500 reliability prediction} \
    -te_type flash \
    -source sn_29500 \
```

```
    -fr_permanent 12.0 \
    -fr_transient 9.0 \
    -fr_transient_set 0.0 \
    -unit_design_element_size_permanent 1.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0 \
    -user_defined_arguments {}
```

5.6.4 create_sm

Table 7—create_sm command

Purpose	Create a safety mechanism	
Syntax	**create_sm** *sm_name* [**-sm_description** *string*] [**-parent_fmeda** *parent_fmeda*] [**-class** <**hw** | **sw** | **aou** | **aou-sw** | **aou-hw** | **user-defined** >] [**-class_description** *string*] **-configurable** [< **TRUE** | **FALSE** >] **-dc_permanent** *float* [*0.0*, *100.0*] **-dc_transient** *float* [*0.0*, *100.0*] **-dc_latent_primary** *float* [*0.0*, *100.0*] **-dc_latent_secondary** *float* [*0.0*, *100.0*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*sm_name*	The name of the safety mechanism to be created
	-sm_description *string*	A textual description of the safety mechanism
	-parent_fmeda *parent_fmeda*	The FMEDA to which the safety mechanism belongs
	-class <**hw** | **sw** | **aou** | **aou-sw** | **aou-hw** | **user-defined** >	The method by which the safety mechanism is to be realized. AoU (Assumptions of Use) is used when the safety mechanism is not part of the product.
	-class_description *string*	A textual description of the class associated with the safety mechanism
	-configurable [< **TRUE** | **FALSE** >]	Indicates whether the safety mechanism can be enabled or disabled by the user or integrator
	-dc_permanent *float* [*0.0*, *100.0*]	The diagnostic coverage of the safety mechanism in isolation for permanent faults
	-dc_transient *float* [*0.0*, *100.0*]	The diagnostic coverage of the safety mechanism in isolation for transient faults
	-dc_latent_primary *float* [*0.0*, *100.0*]	The diagnostic coverage of the safety mechanism in isolation for latent primary faults. Available when the ASIL target level is defined.
	-dc_latent_secondary *float* [*0.0*, *100.0*]	The diagnostic coverage of the safety mechanism in isolation for latent secondary faults. Available when the ASIL target level is defined.
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_sm command defines a new *safety mechanism* instance for use in FMEDA analysis in accordance with ISO 26262 and IEC 61508.

- *sm_name* — specifies the unique name assigned to the *safety mechanism*.
- **-sm_description** — provides an optional textual description of the *safety mechanism* to enhance documentation and traceability.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *safety mechanism* belongs.
- **-class** — defines the method by which the *safety mechanism* is realized, supporting hw, sw, aou, aou-sw, aou-hw, and user-defined types. AoU (Assumptions of Use) is used to indicate when the *safety mechanism* is not part of the product and may raise a flag during FMEDA integration.
- **-class_description** — provides an optional textual description of the class associated with the *safety mechanism*, particularly useful when the class is user-defined.
- **-configurable** — indicates whether the *safety mechanism* can be enabled or disabled by the user or integrator. If **create_sm -configurable** is set to TRUE, then **assign_sm_fm -active** can be specified as TRUE or FALSE to control the activation state of the *safety mechanism* for a given *failure mode*.
- **-dc_permanent** and **-dc_transient** — specify the diagnostic coverage of the *safety mechanism* in isolation for permanent and transient faults, respectively, each expressed as a percentage in the range [0.0, 100.0]. These values quantify the effectiveness of the *safety mechanism* in detecting or mitigating permanent and transient faults independently. These fields correspond to K_{RF} in Figure 10 of ISO 26262-2018, Part 10.
- **-dc_latent_primary** and **-dc_latent_secondary** — specify the diagnostic coverage of the *safety mechanism* in isolation for latent primary and latent secondary faults, respectively, each expressed as a percentage in the range [0.0, 100.0]. These fields are available when the ASIL target level is defined and correspond to $K_{MPF,primary}$ and $K_{MPF,secondary}$ in Figure 10 of ISO 26262-2018, Part 10.

Rules

The following rules apply:

Table 8—Rules for ‘create_sm’ command

Rule ID	Rule
Rule_SM_1 (Inherited from Rule_FMEDA_7)	It shall be an error to execute the **create_sm** command on an FMEDA referenced by the **-parent_fmeda** argument if that FMEDA was created using the **-master_fmeda** option. Instantiated FMEDAs shall not permit the creation of new safety mechanisms.
Rule_SM_2	When **-configurable** is set to TRUE in **create_sm**, the argument **-active** shall be available in **assign_sm_fm**.
Rule_SM_3 (Inherited from Rule_FMEDA_3)	It shall be an error if **-dc_latent_primary** is specified in **create_sm** when the ASIL level of the associated *FMEDA* is set to none (i.e., **-asil** is none in **create_fmeda**).
Rule_SM_4 (Inherited from Rule_FMEDA_3)	It shall be an error if **-dc_latent_secondary** is specified in **create_sm** when the ASIL level of the associated *FMEDA* is set to none (i.e., **-asil** is none in **create_fmeda**).

How-to Examples

Example 1: Software-based Watchdog Timer Safety Mechanism (Configurable, ASIL Context)

This example creates a software watchdog timer safety mechanism that monitors system execution and detects stuck-at faults, with configurable activation for different operation modes. Includes latent DC values for ASIL-based FMEDA context.

```
create_sm SW_Watchdog_Timer \
```

```
    -sm_description {Software watchdog timer for deadlock and stuck-at fault detection} \
    -parent_fmeda MCU_FMEDA \
    -class sw \
    -class_description {Periodic timer with software-based monitoring and reset capability} \
    -configurable TRUE \
    -dc_permanent 80.0 \
    -dc_transient 60.0 \
    -dc_latent_primary 75.0 \
    -dc_latent_secondary 50.0 \
    -user_defined_arguments {}
```

Example 2: Hybrid AoU-HW Safety Mechanism - CRC Check (Configurable)

This example creates a hybrid AoU-HW safety mechanism where CRC checking hardware is present in the product but activation requires external configuration. Demonstrates Rule_SM_1 with configurable=TRUE.

```
create_sm CRC_Data_Integrity \
    -sm_description {Hardware CRC checker requiring external configuration for activation} \
    -parent_fmeda Communication_FMEDA \
    -class aou-hw \
    -class_description {On-chip CRC engine with assumed external configuration by integrator} \
    -configurable TRUE \
    -dc_permanent 98.0 \
    -dc_transient 95.0 \
    -dc_latent_primary 92.0 \
    -dc_latent_secondary 85.0 \
    -user_defined_arguments {}
```

Example 3: User-defined Class - Custom Parity Check

This example creates a user-defined safety mechanism class for a custom parity checking implementation specific to the design.

```
create_sm Custom_Parity_Check \
    -sm_description {Application-specific parity checking for critical data paths} \
    -parent_fmeda Data_Path_FMEDA \
    -class user-defined \
    -class_description {Custom triple-modular parity with majority voting for critical registers} \
    -configurable FALSE \
    -dc_permanent 85.0 \
    -dc_transient 80.0 \
    -dc_latent_primary 75.0 \
    -dc_latent_secondary 60.0 \
    -user_defined_arguments {}
```

Example 4: AoU-SW Safety Mechanism - External Software Monitor (Non-ASIL Context)

This example creates an AoU-SW safety mechanism where software monitoring is assumed to be provided externally. Demonstrates usage without latent DC values when ASIL is not defined (Rule_SM_2, Rule_SM_3).

```
create_sm External_SW_Monitor \
```

```
    -sm_description {External software health monitoring assumed by system integrator} \
    -parent_fmeda IEC61508_FMEDA \
    -class aou-sw \
    -class_description {Assumes external software-based health monitoring and recovery} \
    -configurable FALSE \
    -dc_permanent 65.0 \
    -dc_transient 55.0 \
    -dc_latent_primary {} \
    -dc_latent_secondary {} \
    -user_defined_arguments {}
```

Example 5: Hardware ECC Safety Mechanism (Non-ASIL Context)

This example creates a hardware ECC safety mechanism for IEC 61508 context where ASIL is not defined, demonstrating proper usage without latent DC values per Rule_SM_2 and Rule_SM_3.

```
create_sm ECC_Memory_Protection \
    -sm_description {Hardware-based error correction code for SRAM in industrial SIL context} \
    -parent_fmeda Industrial_FMEDA \
    -class hw \
    -class_description {Dedicated ECC logic with single-error correction and double-error detection}
\
    -configurable FALSE \
    -dc_permanent 95.0 \
    -dc_transient 90.0 \
    -dc_latent_primary {} \
    -dc_latent_secondary {} \
    -user_defined_arguments {}
```

5.6.5 create_element

Table 9—create_element command

Purpose	Create an element to be included in the functional safety hierarchy	
Syntax	**create_element** *element_name* [**-element_description** *string*] **-element_type** <**component** | **subcomponent** | **part** | **subpart** > [**-parent_element** *parent_element*] **-parent_fmeda** *parent_fmeda* [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*element_name*	The name of the element to be created
	-element_description *string*	A textual description of the intended functionality of the element
	-element_type <**component** | **subcomponent** | **part** | **subpart** >	The type of the element being defined. The value component or subcomponent is used for FMEDA defined according to IEC 61508. The value part or subpart is used for FMEDA defined according to ISO 26262
	-parent_element *parent_element*	The identifier of the parent element to which this element is connected in the FS hierarchy
	-parent_fmeda *parent_fmeda*	The name of the FMEDA to which the element belongs

	-user_defined_arguments A list of user-defined arguments and their values *list_of_tuples*
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.

create_element command defines a new *element* to be included in the *FS hierarchy* for FMEDA analysis in accordance with ISO 26262 and IEC 61508. This **create_element** command supports the creation of both flat and hierarchical *FMEDA*.

- *element_name* — specifies the unique name assigned to the *element*.
- **-element_description** — provides an optional textual description of the intended functionality of the *element* to enhance documentation and traceability.
- **-element_type** — defines the type of the *element*, supporting component and subcomponent for *FMEDA* defined according to IEC 61508, and part and subpart for *FMEDA* defined according to ISO 26262. This ensures correct alignment with the requirements of the respective standard.
- **-parent_element** — specifies the identifier of the parent *element* to which this *element* is connected in the *FS hierarchy*, enabling unique identification of the *element*.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *element* belongs.

Rules

The following rules apply:

Table 10—Rules for 'create_element' command

Rule ID	Rule
Rule_Element_1 (Inherited from Rule_FMEDA_7)	It shall be an error to execute the **create_element** command on an FMEDA referenced by the **-parent_fmeda** argument if that FMEDA was created using the **-master_fmeda** option. Instantiated FMEDAs shall not permit the creation of new elements.
Rule_Element_2	It shall be an error if the **-parent_element** argument is used on an element that already contains child failure modes.
Rule_Element_3	It shall be an error if the FMEDA referenced by the **-parent_fmeda** argument is not a leaf in the FMEDA hierarchy (a leaf FMEDA is defined as an FMEDA that has no child FMEDA).
Rule_Element_4	It shall be an error if an *element* of **-element_type** part has a direct or indirect parent of **-element_type** part.
Rule_Element_5	It shall be an error if an *element* of **-element_type** subpart does not have a direct or indirect parent of **-element_type** part.
Rule_Element_6 (Inherited from Rule_FMEDA_3)	It shall be an error if this command does not comply with the FMEDA-level semantic requirements defined in Rule_FMEDA_3, or equivalently it shall be an error if in **create_fmeda -asil** is defined as a, b, c, or d and **-sil** is none, while in **create_element -element_type** is component or subcomponent.
Rule_Element_7 (Inherited from Rule_FMEDA_4)	It shall be an error if this command does not comply with the FMEDA-level semantic requirements defined in Rule_FMEDA_4, or equivalently it shall be an error if in **create_fmeda -sil** is defined as 1, 2, 3, or 4 and **-asil** is none, while in **create_element -element_type** is part or subpart.

How-to Examples

Example 1: Top-Level Part Element for ISO 26262 Automotive SoC

This example creates a top-level part element representing the main SoC in an ISO 26262 FMEDA hierarchy, with no parent element.

```
create_element Automotive_SoC \
    -element_description {Main system-on-chip for automotive electronic control unit} \
    -element_type part \
    -parent_element {} \
    -parent_fmeda ECU_FMEDA \
    -user_defined_arguments {}
```

Example 2: Subpart Element for ISO 26262 CPU Core

This example creates a subpart element representing a CPU core within an automotive SoC, demonstrating hierarchical FMEDA structure for ISO 26262 compliance.

```
create_element CPU_Core \
    -element_description {Dual-core ARM processor for safety-critical computation} \
    -element_type subpart \
    -parent_element Automotive_SoC \
    -parent_fmeda ECU_FMEDA \
    -user_defined_arguments {}
```

Example 3: Component Element for IEC 61508 Industrial Controller

This example creates a component element for an IEC 61508 industrial safety controller, showing proper element_type for IEC standard compliance.

```
create_element PLC_Controller \
    -element_description {Programmable logic controller for industrial process safety} \
    -element_type component \
    -parent_element {} \
    -parent_fmeda Industrial_PLC_FMEDA \
    -user_defined_arguments {}
```

Example 4: Subcomponent Element for IEC 61508 Memory Subsystem

This example creates a subcomponent element representing a memory subsystem within an industrial controller, demonstrating hierarchical structure for IEC 61508 compliance.

```
create_element SRAM_Controller \
    -element_description {Safety-critical SRAM controller with ECC protection} \
    -element_type subcomponent \
    -parent_element PLC_Controller \
    -parent_fmeda Industrial_PLC_FMEDA \
    -user_defined_arguments {}
```

Example 5: Nested Subpart Element for ISO 26262 ALU

This example creates a nested subpart element representing an ALU within a CPU core, demonstrating multi-level subpart hierarchy allowed by Rule_Element_2 for ISO 26262 compliance.

```
create_element Arithmetic_Logic_Unit \
    -element_description {Integer and floating-point arithmetic logic unit} \
    -element_type subpart \
```

```
    -parent_element CPU_Core \
    -parent_fmeda ECU_FMEDA \
    -user_defined_arguments {}
```

5.6.6 create_fm

Table 11—create_fm command

Purpose	Create a failure mode to be included in the failure mode hierarchy	
Syntax	**create_fm** *fm_name* [**-fm_description** *string*] **-parent_element** *parent_element* **-parent_fmeda** *parent_fmeda* **-safety_relevant** [< **TRUE** | **FALSE** >] **-no_part** [< **TRUE** | **FALSE** >] **-safeness_permanent_estimated** *float* [*0.0*, *100.0*] **-safeness_transient_estimated** *float* [*0.0*, *100.0*] [**-safeness_permanent_measured** *float* [*0.0*, *100.0*]] [**-safeness_transient_measured** *float* [*0.0*, *100.0*]] **-pvsg_permanent** *float* [*0.0*, *100.0*] **-pvsg_transient** *float* [*0.0*, *100.0*] [**-dc_permanent_expert** *float* [*0.0*, *100.0*]] [**-dc_transient_expert** *float* [*0.0*, *100.0*]] [**-dc_latent_primary_expert** *float* [*0.0*, *100.0*]] [**-dc_latent_secondary_expert** *float* [*0.0*, *100.0*]] [**-dc_permanent_measured** *float* [*0.0*, *100.0*]] [**-dc_transient_measured** *float* [*0.0*, *100.0*]] [**-dc_latent_primary_measured** *float* [*0.0*, *100.0*]] [**-dc_latent_secondary_measured** *float* [*0.0*, *100.0*]] **-perceived_primary_permanent** *float* [*0.0*, *100.0*] **-perceived_primary_transient** *float* [*0.0*, *100.0*] **-perceived_secondary_permanent** *float* [*0.0*, *100.0*] **-perceived_secondary_transient** *float* [*0.0*, *100.0*] **-no_effect_permanent** *float* [*0.0*, *100.0*] **-no_effect_transient** *float* [*0.0*, *100.0*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*fm_name*	The name of the failure mode to be created
	-fm_description *string*	The description of the failure mode
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the failure mode
	-parent_fmeda *parent_fmeda*	The FMEDA associated with the failure mode
	-safety_relevant [< **TRUE** | **FALSE** >]	Specifies whether the failure mode is safety-relevant in the context of ISO 26262. *This argument is required only for ISO 26262 (i.e., if the FMEDA has the ASIL level defined).
	-no_part [< **TRUE** | **FALSE** >]	Specifies whether the failure mode is classified as having no part in a dangerous failure in the context of IEC 61508. When set to TRUE, the failure mode does not contribute to a dangerous condition *This argument is required only for IEC 61508 (i.e., if the FMEDA has the SIL level defined).
	-safeness_permanent_estimated *float* [*0.0*, *100.0*]	The estimated percentage of safeness for permanent faults
	-safeness_transient_estimated *float* [*0.0*, *100.0*]	The estimated percentage of safeness for transient faults
	-safeness_permanent_measured *float* [*0.0*, *100.0*]	The measured percentage of safeness for permanent faults

	-safeness_transient_measured *float* [*0.0*, *100.0*]	The measured percentage of safeness for transient faults
	-pvsg_permanent *float* [*0.0*, *100.0*]	The percentage of permanent faults with potential to violate a safety goal in the absence of a safety mechanism
	-pvsg_transient *float* [*0.0*, *100.0*]	The percentage of transient faults with potential to violate a safety goal in the absence of a safety mechanism
	-dc_permanent_expert *float* [*0.0*, *100.0*]	The diagnostic coverage for permanent faults specified by expert, aggregated over the safety mechanisms associated with the failure mode
	-dc_transient_expert *float* [*0.0*, *100.0*]	The diagnostic coverage for transient faults specified by expert, aggregated over the safety mechanisms associated with the failure mode
	-dc_latent_primary_expert *float* [*0.0*, *100.0*]	The diagnostic coverage for latent primary faults specified by expert, aggregated over the safety mechanisms associated with the failure mode
	-dc_latent_secondary_expert *float* [*0.0*, *100.0*]	The diagnostic coverage for latent secondary faults specified by expert, aggregated over the safety mechanisms associated with the failure mode
	-dc_permanent_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage for permanent faults specified as a result of fault injection activities, aggregated over the safety mechanisms associated with the failure mode
	-dc_transient_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage for transient faults specified as a result of fault injection activities, aggregated over the safety mechanisms associated with the failure mode
	-dc_latent_primary_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage for latent primary faults specified as a result of fault injection activities, aggregated over the safety mechanisms associated with the failure mode
	-dc_latent_secondary_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage for latent secondary faults specified as a result of fault injection activities, aggregated over the safety mechanisms associated with the failure mode
	-perceived_primary_permanent *float* [*0.0*, *100.0*]	The percentage of permanent primary multi-point faults that are not detected but are perceived. *This argument is required only for ISO 26262 (i.e., if the FMEDA has the ASIL level defined).
	-perceived_primary_transient *float* [*0.0*, *100.0*]	The percentage of transient primary multi-point faults that are not detected but are perceived. *This argument is required only for ISO 26262 (i.e., if the FMEDA has the ASIL level defined).
	-perceived_secondary_permanent *float* [*0.0*, *100.0*]	The percentage of permanent secondary multi-point faults that are not detected but are perceived. *This argument is required only for ISO 26262 (i.e., if the FMEDA has the ASIL level defined).
	-perceived_secondary_transient *float* [*0.0*, *100.0*]	The percentage of transient secondary multi-point faults that are not detected but are perceived. *This argument is required only for ISO 26262 (i.e., if the FMEDA has the ASIL level defined).
	-no_effect_permanent *float* [*0.0*, *100.0*]	The percentage of permanent no effect faults. *This argument is required only for IEC 61508 (i.e., if the FMEDA has the SIL level defined).
	-no_effect_transient *float* [*0.0*, *100.0*]	The percentage of transient no effect faults. *This argument is required only for IEC 61508 (i.e., if the FMEDA has the SIL level defined).

	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_fm command defines a new *failure mode* in the *FM hierarchy* within a specified *FMEDA*, supporting both ISO 26262 and IEC 61508.

- *fm_name* — specifies the name of the *failure mode* to be created in the *FM hierarchy*.
- **-fm_description** — provides an optional textual description of the *failure mode*.
- **-parent_element** — specifies the identifier of the parent *element* to which the *failure mode* is connected in the *FS hierarchy*, enabling unique identification of the *failure mode*.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *failure mode* belongs.
- **-safety_relevant** — indicates whether the *failure mode* is safety-relevant in the context of ISO 26262. This field is mandatory if the *FMEDA* has an ASIL level defined.
- **-no_part** — indicates whether the *failure mode* is classified as having no part in a dangerous failure in the context of IEC 61508. When set to TRUE, the *failure mode* does not contribute to a dangerous condition. This field is mandatory if the *FMEDA* has a SIL level defined.
- **-safeness_permanent_estimated** and **-safeness_transient_estimated** — specify the estimated percentage [0.0, 100.0] of safeness for permanent and transient faults (i.e., faults that do not contribute to the violation of a *safety goal*), respectively.
- **-safeness_permanent_measured** and **-safeness_transient_measured** — specify the measured percentage [0.0, 100.0] of safeness for permanent and transient faults, respectively.
- **-pvsg_permanent** and **-pvsg_transient** — specify the percentage [0.0, 100.0] of permanent and transient faults, respectively, with potential to violate a *safety goal* in the absence of a *safety mechanism*.
- **-dc_permanent_expert**, **-dc_transient_expert**, **-dc_latent_primary_expert**, and **-dc_latent_secondary_expert** — specify the diagnostic coverage for permanent, transient, latent primary, and latent secondary faults, respectively, as specified by expert judgment aggregated over the *safety mechanisms* associated with the *failure mode*.
- **-dc_permanent_measured**, **-dc_transient_measured**, **-dc_latent_primary_measured**, and **-dc_latent_secondary_measured** — specify the measured diagnostic coverage for permanent, transient, latent primary, and latent secondary faults, respectively, as a result of fault injection or verification activities, aggregated over the *safety mechanisms* associated with the *failure mode*.
- **-perceived_primary_permanent**, **-perceived_primary_transient**, **-perceived_secondary_permanent**, and **-perceived_secondary_transient** — specify the percentage [0.0, 100.0] of permanent and transient primary and secondary multi-point faults that are not detected but are perceived. These fields are mandatory if the *FMEDA* has an ASIL level defined.
- **-no_effect_permanent** and **-no_effect_transient** — specify the percentage [0.0, 100.0] of permanent and transient faults, respectively, with no effect, as defined by IEC 61508. These fields are mandatory if the *FMEDA* has a SIL level defined.

Rules

The following rules apply:

Table 12—Rules for ‘create_fm’ command

Rule ID	Rule
Rule_FM_1 (Inherited from Rule_FMEDA_7)	It shall be an error to execute the **create_fm** command on an FMEDA referenced by the **-parent_fmeda** argument if that FMEDA was created using the **-master_fmeda** option. Instantiated FMEDAs shall not permit the creation of new failure modes.
Rule_FM_2 (Inherited from Rule_FMEDA_3)	It shall be an error if in **create_fmeda -asil** is defined as a, b, c, or d and **-sil** is none, while **create_fm -no_effect_*** is used.
Rule_FM_3 (Inherited from Rule_FMEDA_4)	It shall be an error if in **create_fmeda -sil** is defined as 1, 2, 3, or 4 and **-asil** is none, while **create_fm -perceived_***, **-pvsg_***, **-safety_relevant**, **-dc_latent_*** are used.
Rule_FM_4	It shall be an error if **create_fm** is used on an *element* that is not a leaf in the *FS hierarchy* (a leaf *element* is defined as an *element* that has no child *elements*, i.e., no child subparts or subcomponents).

How-to Examples

Example 1: Safety-Relevant FM for ISO 26262 Automotive Microcontroller

This example creates a safety-relevant failure mode for an arithmetic logic unit (ALU) in an automotive microcontroller, demonstrating ISO 26262-specific arguments.

```
create_fm ALU_Stuck_At_Fault \
        -fm_description {Stuck-at fault in ALU causing incorrect computation} \
        -parent_element ALU_Core \
        -parent_fmeda MCU_Safety_Analysis \
        -safety_relevant TRUE \
        -no_part {} \
        -safeness_permanent_estimated 5.0 \
        -safeness_transient_estimated 2.0 \
        -pvsg_permanent 85.0 \
        -pvsg_transient 75.0 \
        -dc_latent_primary_expert 90.0 \
        -perceived_primary_permanent 8.0 \
        -perceived_primary_transient 5.0 \
        -perceived_secondary_permanent 2.0 \
        -perceived_secondary_transient 1.0 \
        -no_effect_permanent {} \
        -no_effect_transient {} \
        -user_defined_arguments {}
```

Example 2: Non-Safety-Relevant FM for ISO 26262 SoC

This example demonstrates a non-safety-relevant failure mode in an ISO 26262 system-on-chip, showing proper use of safety_relevant FALSE.

```
create_fm Debug_Port_Failure \
```

```
        -fm_description {Failure in debug interface with no safety impact} \
        -parent_element Debug_Interface \
        -parent_fmeda SoC_FMEDA \
        -safety_relevant FALSE \
        -no_part {} \
        -safeness_permanent_estimated 0.0 \
        -safeness_transient_estimated 0.0 \
        -pvsg_permanent {} \
        -pvsg_transient {} \
        -perceived_primary_permanent 0.0 \
        -perceived_primary_transient 0.0 \
        -perceived_secondary_permanent 0.0 \
        -perceived_secondary_transient 0.0 \
        -no_effect_permanent {} \
        -no_effect_transient {} \
        -user_defined_arguments {}
```

Example 3: ISO 26262 FM with Measured Diagnostic Coverage

This example demonstrates an ISO 26262 failure mode with measured diagnostic coverage values from fault injection testing, showing both expert and measured DC values.

```
create_fm Clock_Domain_Crossing_Error \
        -fm_description {Metastability fault in clock domain crossing circuit} \
        -parent_element Clock_Management \
        -parent_fmeda FPGA_Safety_FMEDA \
        -safety_relevant TRUE \
        -no_part {} \
        -safeness_permanent_estimated 3.0 \
        -safeness_transient_estimated 12.0 \
        -safeness_permanent_measured 2.5 \
        -safeness_transient_measured 10.5 \
        -pvsg_permanent 80.0 \
        -pvsg_transient 88.0 \
        -dc_latent_primary_expert 85.0 \
        -dc_latent_secondary_expert 60.0 \
        -dc_latent_primary_measured 87.5 \
        -dc_latent_secondary_measured 62.0 \
        -perceived_primary_permanent 6.0 \
        -perceived_primary_transient 4.0 \
        -perceived_secondary_permanent 3.0 \
        -perceived_secondary_transient 2.5 \
        -no_effect_permanent {} \
        -no_effect_transient {} \
        -user_defined_arguments {}
```

Example 4: IEC 61508 Sensor Interface FM with Measured Values

This example creates a failure mode for an IEC 61508 sensor interface in a process control system, demonstrating both measured safeness and measured diagnostic coverage values from fault injection testing.

```
create_fm ADC_Conversion_Error \
        -fm_description {Analog-to-digital conversion error in temperature sensor} \
```

```
        -parent_element Temperature_Sensor_ADC \
        -parent_fmeda Process_Control_FMEDA \
        -safety_relevant {} \
        -no_part TRUE \
        -safeness_permanent_estimated 15.0 \
        -safeness_transient_estimated 20.0 \
        -safeness_permanent_measured 14.2 \
        -safeness_transient_measured 18.5 \
        -pvsg_permanent {} \
        -pvsg_transient {} \
        -dc_permanent_expert 80.0 \
        -dc_transient_expert 75.0 \
        -dc_permanent_measured 82.5 \
        -dc_transient_measured 77.0 \
        -dc_latent_primary_expert {} \
        -dc_latent_secondary_expert {} \
        -dc_latent_primary_measured {} \
        -dc_latent_secondary_measured {} \
        -perceived_primary_permanent {} \
        -perceived_primary_transient {} \
        -perceived_secondary_permanent {} \
        -perceived_secondary_transient {} \
        -no_effect_permanent 12.0 \
        -no_effect_transient 8.0 \
        -user_defined_arguments {}
```

5.6.7 create_fme

Table 13—create_fme command

Purpose	Create a failure mode effect	
Syntax	**create_fme** *fme_name* [**-fme_description** *string*] **-parent_fmeda** *parent_fmeda* [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*fme_name*	The name of the failure mode effect to be created
	-fme_description *string*	The description of the failure mode effect
	-parent_fmeda *parent_fmeda*	The FMEDA associated with the failure mode effect
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_fme command defines a new *failure mode effect* instance for use in FMEDA analysis.

- *fme_name* — specifies the unique name assigned to the *failure mode effect*.
- **-fme_description** — provides an optional textual description of the *failure mode effect* to enhance documentation and traceability.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *failure mode effect* belongs.

Failure mode effects enable the grouping, abstraction, and transfer of failure rate contributions from one or more *failure modes* within a safety analysis to a higher-level safety analysis scope.

Rules

The following rules apply:

Table 14—Rules for 'create_fme' command

Rule ID	Rule
Rule_FME_1 (Inherited from Rule_FMEDA_7)	It shall be an error to execute the **create_fme** command on an FMEDA referenced by the **-parent_fmeda** argument if that FMEDA was created using the **-master_fmeda** option. Instantiated FMEDAs shall not permit the creation of new failure mode effects.
Rule_FME_2	Each *failure mode effect* defined by **create_fme** shall be non-overlapping with all other *failure mode effects* associated with the same *FMEDA*.

How-to Examples

Example 1: Loss of Function Effect for Safety-Critical Computation

This example creates a failure mode effect representing complete loss of arithmetic computation function, grouped from multiple ALU failure modes for ISO 26262 safety analysis.

```
create_fme Loss_Of_Arithmetic_Function \
    -fme_description {Complete loss of ALU arithmetic computation capability affecting safety-
critical calculations} \
    -parent_fmeda CPU_Core_FMEDA \
    -user_defined_arguments {}
```

Example 2: Erroneous Function Effect for Memory Read Operations

This example creates a failure mode effect representing erroneous memory read function where data is corrupted, grouped from memory controller failure modes for IEC 61508 compliance.

```
create_fme Erroneous_Memory_Read_Function \
    -fme_description {Memory read function returns incorrect data due to bit flips or addressing
errors} \
    -parent_fmeda Memory_Controller_FMEDA \
    -user_defined_arguments {}
```

5.6.8 create_sreq

Table 15—create_sreq command

Purpose	Create a safety requirement of type safety goal or top-level safety requirement	
Syntax	**create_sreq** *sreq_name* **-sreq_type** <**safety_goal** | **top_level_safety_requirement** > [**-sreq_description** *string*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	*sreq_name*	The name of the safety requirement
	-sreq_type <**safety_goal** | **top_level_safety_requirement** >	The type of the safety requirement; either safety_goal or top_level_safety_requirement
	-sreq_description *string*	The description of the safety requirement

	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

create_sreq command defines a new *safety requirement*.

- *sreq_name* — specifies the unique name of the *safety requirement* to be created.
- **-sreq_type** — specifies the type of the *safety requirement* and shall be set to either safety_goal or top_level_safety_requirement.
- **-sreq_description** — provides an optional textual description of the *safety requirement*, supporting traceability and documentation.

Safety requirements define high-level safety objectives for a design and enable grouping of *failure modes* for SoC-level safety metrics reporting. The term *safety requirements* includes both *safety goals* and top-level safety requirements, aligning with common industry practice.

How-to Examples

Example 1: Safety Goal for ISO 26262 Automotive SoC

This example creates a safety goal for an automotive microcontroller FMEDA, defining a high-level safety objective for ASIL D compliance.

```
create_sreq Prevent_Unintended_Acceleration \
    -sreq_type safety_goal \
    -sreq_description {Safety goal to prevent unintended vehicle acceleration due to ECU failures} \
    -user_defined_arguments {}
```

Example 2: Top-Level Safety Requirement for IEC 61508 Industrial Controller

This example creates a top-level safety requirement for an industrial process control system targeting SIL 3, grouping failure modes for safety metrics reporting.

```
create_sreq Emergency_Shutdown_Protection \
    -sreq_type top_level_safety_requirement \
    -sreq_description {Top-level requirement ensuring emergency shutdown capability under all fault
conditions} \
    -user_defined_arguments {}
```

Example 3: Safety Goal for Automotive Powertrain Control Unit (ISO 26262 ASIL C)

This example defines a safety goal for an automotive powertrain control unit SoC, targeting ASIL C compliance for engine management safety functions.

```
create_sreq Prevent_Engine_Overspeed \
    -sreq_type safety_goal \
    -sreq_description {Safety goal to prevent engine overspeed conditions due to powertrain ECU
hardware failures} \
    -user_defined_arguments {}
```

5.6.9 assign_te_element

Table 16—assign_te_element command

Purpose	Assign a technology element to an element in the functional safety hierarchy	
Syntax	**assign_te_element** **-te_name** *te_name* **-element_name** *element_name* **-parent_element** *parent_element* **-parent_fmeda** *parent_fmeda* [**-element_size_permanent** *float* [*0.0*, *n*]] [**-element_size_transient** *float* [*0.0*, *n*]] [**-element_size_transient_set** *float* [*0.0*, *n*]] [**-element_mapping_format** <**sv** | **vhdl** | **spice** | **user-defined** >] [**-element_mapping** *element_mapping_list*] [**-element_mapping_exclude** *element_mapping_exclude_list*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-te_name** *te_name*	The name of the technology element in which the element is implemented
	-element_name *element_name*	The name of the element in the FS hierarchy to which the technology element is assigned
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the element
	-parent_fmeda *parent_fmeda*	The FMEDA associated with the assignment
	-element_size_permanent *float* [*0.0*, *n*]	The size of the element for permanent faults for the associated technology element; this value takes precedence for assumption-based FMEDA, otherwise the size is calculated based on the area extracted by the mapping to the design hierarchy. *Mandatory/ optional status is defined by the reference table below
	-element_size_transient *float* [*0.0*, *n*]	The size of the element for transient faults for the associated technology element; this value takes precedence for assumption-based FMEDA, otherwise the size is calculated based on the area extracted by the mapping to the design hierarchy. *Mandatory/ optional status is defined by the reference table below
	-element_size_transient_set *float* [*0.0*, *n*]	The size of the element for single event transient faults for the associated technology element; this value takes precedence for assumption-based FMEDA, otherwise the size is calculated based on the area extracted by the mapping to the design hierarchy. *Mandatory/optional status is defined by the reference table below
	-element_mapping_format <**sv** | **vhdl** | **spice** | **user-defined** >	The format of the design hierarchy mapping
	-element_mapping *element_mapping_list*	The design representation identifying the portion of the design implementing the intended functionality of the element; this value takes precedence for calculation-based FMEDA *Mandatory/ optional status is defined by the reference table below
	-element_mapping_exclude *element_mapping_exclude_list*	The design representation identifying the portion of the design to be excluded from the element mapping; this value is used only in conjunction with -element_mapping for calculation-based FMEDA
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

Table 17—Argument-dependency conformance rules for the assign_te_element and assign_te_fm commands

Argument	-fm_size_type_assumption_based			-fm_size_type_calculation_based		
	absolute	percentage	uniform_ distribution	mapping_ scaling	mapping_ percentage	mapping_ uniform
-element_size _permanent	-	R	R	-	-	-
-element_size _transient	-	R	R	-	-	-
-element_size _transient_set	-	R	R	-	-	-
-element_mapping	-	-	-	-	R	R
R = Required, - = Not Applicable						

assign_te_element command assigns a *technology element* to an *element* in the *FS hierarchy* within a specified *FMEDA*, supporting both assumption-based and calculation-based FMEDA methodologies.

- **-te_name** — specifies the name of the *technology element* in which the *element* is implemented.
- **-element_name** — specifies the name of the *element* in the *FS hierarchy* to which the *technology element* is assigned.
- **-parent_element** — specifies the identifier of the parent *element* to which this *element* is connected in the *FS hierarchy*, enabling unique identification of the *element*.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *element* belongs.

The command enables the definition of *element* size either directly or by mapping.

- **-element_size_permanent**, **-element_size_transient**, and **-element_size_transient_set** — specify the size [unitless, float ≥ 0] of the *element* for permanent, transient, and single event transient faults, respectively, for the associated *technology element*. These values take precedence for assumption-based FMEDA; otherwise, the size is calculated based on the area extracted by the mapping to the design hierarchy. In which cases **-element_size_*** is mandatory is defined by the mandatory/optional reference Table 17.

Mapping arguments take precedence for calculation-based FMEDA:

- **-element_mapping_format** — specifies the format of the design hierarchy mapping and may be set to sv, vhdl, spice, or user-defined.
- **-element_mapping** — defines the design representation identifying the portion of the design responsible for the *element*. In which cases **-element_mapping** is mandatory is defined by the mandatory/optional reference Table 17.
- **-element_mapping_exclude** — specifies the portion of the design hierarchy to be excluded from the *element* mapping and is only used in conjunction with **-element_mapping** for calculation-based FMEDA.

Rules

The following rules apply:

Table 18—Rules for 'assign_te_element' command

Rule ID	Rule
Rule_TE_Element_1	It shall be an error if neither the **-element_size_*** nor the **-element_mapping** argument is specified.
Rule_TE_Element_2	It shall be an error if the required/optional status defined by the reference table is not satisfied: -The argument **-element_size_*** is required when the value of the **-fm_size_type_assumption_based** argument in the **assign_te_fm** command is percentage or uniform_distribution. -The argument **-element_mapping** is required when the value of the **-fm_size_type_calculation_based** argument in the **assign_te_fm** command is mapping_percentage or mapping_uniform_distribution.
Rule_TE_Element_3	It shall be an error if **-element_mapping_exclude** is specified without also specifying **-element_mapping**.

How-to Examples

Prerequisite Examples

These examples show the prerequisite commands needed before using assign_te_element.

Prerequisite: Create Technology Element (TE)

```
create_te CMOS_28nm_Logic \
    -te_type digital \
    -fr_permanent 10.0 \
    -fr_transient 5.0 \
    -fr_transient_set 2.0 \
    -unit_design_element_size_permanent 1.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0
```

Prerequisite: Create Element in FS Hierarchy (Top-Level Part)

```
create_element ALU_Part \
    -element_type part \
    -parent_fmeda SoC_FMEDA

create_element Register_Subpart \
    -element_type subpart \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA

create_te SRAM_28nm \
    -te_type ram \
    -fr_permanent 8.0 \
    -fr_transient 8.0 \
    -fr_transient_set 0.0 \
```

```
    -unit_design_element_size_permanent 1.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0

create_element Cache_Part \
    -element_type part \
    -parent_fmeda SoC_FMEDA
```

Assignment Examples

Example 1: Basic Assumption-Based FMEDA with Element Size

This example shows a basic assignment of a technology element to a top-level part in the FS hierarchy using assumption-based FMEDA methodology where element sizes are directly specified.

```
assign_te_element \
    -te_name CMOS_28nm_Logic \
    -element_name ALU_Part \
    -parent_element {} \
    -parent_fmeda SoC_FMEDA \
    -element_size_permanent 5000.0 \
    -element_size_transient 5000.0 \
    -element_size_transient_set 5000.0 \
    -element_mapping_format {} \
    -element_mapping {} \
    -element_mapping_exclude {} \
    -user_defined_arguments {}
```

Example 2: Calculation-Based FMEDA with Design Hierarchy Mapping

This example demonstrates calculation-based FMEDA where the element size is calculated based on design hierarchy mapping. It shows a hierarchical FS structure with a subpart element.

```
assign_te_element \
    -te_name CMOS_28nm_Logic \
    -element_name Register_Subpart \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -element_size_permanent 1200.0 \
    -element_size_transient 1200.0 \
    -element_size_transient_set 1200.0 \
    -element_mapping_format sv \
    -element_mapping {top.cpu.alu.register_bank} \
    -element_mapping_exclude {top.cpu.alu.register_bank.test_controller} \
    -user_defined_arguments {}
```

Example 3: Multiple Technology Elements to Same FS Element

This example demonstrates assigning multiple technology elements to the same FS element, which is permitted per Rule_TE_Element_1.

```
assign_te_element \
```

```
    -te_name CMOS_28nm_Logic \
    -element_name Cache_Part \
    -parent_element {} \
    -parent_fmeda SoC_FMEDA \
    -element_size_permanent 2000.0 \
    -element_size_transient 2000.0 \
    -element_size_transient_set 2000.0 \
    -element_mapping_format sv \
    -element_mapping {top.cache.control_logic} \
    -element_mapping_exclude {} \
    -user_defined_arguments {}

assign_te_element \
    -te_name SRAM_28nm \
    -element_name Cache_Part \
    -parent_element {} \
    -parent_fmeda SoC_FMEDA \
    -element_size_permanent 8000.0 \
    -element_size_transient 8000.0 \
    -element_size_transient_set 8000.0 \
    -element_mapping_format sv \
    -element_mapping {top.cache.memory_array} \
    -element_mapping_exclude {} \
    -user_defined_arguments {}
```

5.6.10 assign_te_fm

Table 19—assign_te_fm command

Purpose	Assign a technology element to a failure mode in the failure mode hierarchy	
Syntax	**assign_te_fm** **-te_name** *te_name* **-fm_name** *fm_name* **-parent_element** *parent_element* **-parent_fmeda** *parent_fmeda* **-fm_size_type_assumption_based** <**none** | **absolute** | **percentage** | **uniform_distribution** > **-fm_size_type_calculation_based** <**none** | **mapping_scaling** | **mapping_percentage** | **mapping_uniform** > [**-fm_size_permanent_assumption_based** *float* [*0.0*, *n*]] [**-fm_size_transient_assumption_based** *float* [*0.0*, *n*]] [**-fm_size_transient_set_assumption_based** *float* [*0.0*, *n*]] [**-fm_size_permanent_calculation_based** *float* [*0.0*, *n*]] [**-fm_size_transient_calculation_based** *float* [*0.0*, *n*]] [**-fm_size_transient_set_calculation_based** *float* [*0.0*, *n*]] [**-fm_mapping_format** <**sv** | **vhdl** | **spice** | **user-defined** >] [**-fm_mapping** *fm_mapping_list*] [**-fm_mapping_exclude** *fm_mapping_exclude_list*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-te_name** *te_name*	The name of the technology element in which the failure mode is implemented
	-fm_name *fm_name*	The name of the failure mode in the failure mode hierarchy
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the failure mode

	-parent_fmeda *parent_fmeda*	The FMEDA associated with the assignment
	-fm_size_type_assumption_based <**none** | **absolute** | **percentage** | **uniform_distribution** >	The method for determining the failure mode size for assumption-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_type_calculation_based <**none** | **mapping_scaling** | **mapping_percentage** | **mapping_uniform** >	The method for determining the failure mode size for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_permanent_assumption_based *float* [*0.0*, *n*]	The size of the failure mode for permanent faults for the associated technology element for assumption-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_transient_assumption_based *float* [*0.0*, *n*]	The size of the failure mode for transient faults for the associated technology element for assumption-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_transient_set_assumption_based *float* [*0.0*, *n*]	The size of the failure mode for single event transient faults for the associated technology element for assumption-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_permanent_calculation_based *float* [*0.0*, *n*]	The size of the failure mode for permanent faults for the associated technology element for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_transient_calculation_based *float* [*0.0*, *n*]	The size of the failure mode for transient faults for the associated technology element for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_size_transient_set_calculation_based *float* [*0.0*, *n*]	The size of the failure mode for single event transient faults for the associated technology element for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_mapping_format <**sv** | **vhdl** | **spice** | **user-defined** >	The format of the design hierarchy mapping
	-fm_mapping *fm_mapping_list*	The design representation identifying the portion of the design responsible for the failure mode; this value takes precedence for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-fm_mapping_exclude *fm_mapping_exclude_list*	The design representation identifying the portion of the design to be excluded from the failure mode mapping; this value is used only in conjunction with -fm_mapping for calculation-based FMEDA. *Mandatory/optional status is defined by the reference table below
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

Table 20—Argument-dependency conformance rules for the assign_te_fm command

Argument	-fm_size_type_assumption_based			-fm_size_type_calculation_based		
	absolute	percentage	uniform_ distribution	mapping_ scaling	mapping_ percentage	mapping_ uniform
-fm_size_permanent _assumption_based	R	R	NP	-	-	-
-fm_size_transient _assumption_based	R	R	NP	-	-	-
-fm_size_transient_set _assumption_based	R	R	NP	-	-	-
-fm_size_permanent _calculation_based	-	-	-	R	R	NP
-fm_size_transient _calculation_based	-	-	-	R	R	NP
-fm_size_transient_set _calculation_based	-	-	-	R	R	NP
-fm_mapping	-	-	-	R	NP	NP
R = Required, NP = Not Permitted, - = Not Applicable						

assign_te_fm command assigns a *technology element* to a *failure mode* in the *FM hierarchy* within a specified *FMEDA*, supporting both assumption-based and calculation-based FMEDA methodologies.

- **-te_name** — specifies the name of the *technology element* in which the *failure mode* is implemented.
- **-fm_name** — specifies the name of the *failure mode* in the *FM hierarchy*.
- **-parent_element** — specifies the identifier of the parent *element* to which this *failure mode* is connected in the *FS hierarchy*, enabling unique identification of the *failure mode*.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *failure mode* belongs.

The command enables the definition of *failure mode* size either directly or by mapping.

- **-fm_size_type_assumption_based** — defines the method for determining the *failure mode* size for assumption-based FMEDA. Supported values:
  - a) none: Not used
  - b) absolute: An absolute value in the TE unit
  - c) percentage: A percentage of the parent *element* size (assigned by **assign_te_element -element_size_***)
  - d) uniform_distribution: A uniform distribution of the parent *element* size (assigned by **assign_te_element -element_size_***)
- **-fm_size_type_calculation_based** — defines the method for determining the *failure mode* size for calculation-based FMEDA. Supported values:
  - a) none: Not used
  - b) mapping_scaling: A percentage of the size of the *failure mode* (assigned by **assign_te_fm -fm_mapping**)
  - c) mapping_percentage: A percentage of the size of the parent *element* size (assigned by **assign_te_element -element_mapping**)
  - d) mapping_uniform: A uniform distribution of the parent *element* size (assigned by **assign_te_element -element_mapping**)

Mapping arguments take precedence for calculation-based FMEDA:

- **-fm_size_permanent_assumption_based**, **-fm_size_transient_assumption_based**, and **-fm_size_transient_set_assumption_based** — specify the size [unitless, float ≥ 0] of the *failure mode* for permanent, transient, and single event transient faults, respectively, for assumption-based FMEDA. In which cases they are mandatory is defined by the mandatory/optional reference Table 20.
- **-fm_size_permanent_calculation_based**, **-fm_size_transient_calculation_based**, and **-fm_size_transient_set_calculation_based** — specify the size [unitless, float ≥ 0] of the *failure mode* for permanent, transient, and single event transient faults, respectively, for calculation-based FMEDA. In which cases they are mandatory is defined by the mandatory/optional reference Table 20.
- **-fm_mapping_format** — specifies the format of the design hierarchy mapping and may be set to sv, vhdl, spice, or user-defined.
- **-fm_mapping** — defines the design representation identifying the portion of the design responsible for the *failure mode*. In which cases **-fm_mapping** is mandatory is defined by the mandatory/optional reference Table 20.
- **-fm_mapping_exclude** — specifies the portion of the design hierarchy to be excluded from the *failure mode* mapping and is only used in conjunction with **-fm_mapping** for calculation-based FMEDA.

Rules

The following rules apply:

Table 21—Rules for 'assign_te_fm' command

Rule ID	Rule
Rule_TE_FM_1	It shall be an error if both **-fm_size_type_assumption_based** and **-fm_size_type_calculation_based** are set to none.
Rule_TE_FM_2 (Inherited from Rule_TE_Element_2)	The values percentage and uniform_distribution for **-fm_size_type_assumption_based** are permitted only if the corresponding **-element_size_*** argument of **assign_te_element** is also specified.
Rule_TE_FM_3 (Inherited from Rule_TE_Element_2)	The values mapping_percentage and mapping_uniform_distribution for **-fm_size_type_calculation_based** are permitted only if the corresponding **-element_mapping** argument of **assign_te_element** is also specified.
Rule_TE_FM_4	It shall be an error if the required/optional status defined by the reference table is not satisfied: -The arguments **-fm_size_*_assumption_based** are required if the value of argument **-fm_size_type_assumption_based** is either absolute or percentage. -The arguments **-fm_size_*_assumption_based** are not permitted if the value of the argument **-fm_size_type_assumption_based** is none or uniform_distribution. -The arguments **-fm_size_*_calculation_based** are required if the value of argument **-fm_size_type_calculation_based** is either mapping_scaling or mapping_percentage. -The arguments **-fm_size_*_calculation_based** are not permitted if the value of the argument **-fm_size_type_calculation_based** is mapping_uniform_distribution. -The argument **-fm_mapping** is required if the value of argument **-fm_size_type_calculation_based** is mapping_scaling. -The argument **-fm_mapping** is not permitted if the value of the argument **-fm_size_type_calculation_based** is mapping_percentage or mapping_uniform_distribution.
Rule_TE_FM_5	It shall be an error if **-fm_size_type_assumption_based** is not consistent across all *failure modes* within the same *element*.

Rule_TE_FM_6	It shall be an error if **-fm_size_type_calculation_based** is not consistent across all *failure modes* within the same *element*.
Rule_TE_FM_7	It shall be an error if **-fm_mapping_exclude** is specified without also specifying **-fm_mapping**.

How-to Examples

Prerequisite Examples

These examples show the prerequisite commands needed before using assign_te_fm.

Prerequisite: Create Technology Element (TE)

```
create_te CMOS_28nm_Logic \
    -fr_permanent 10.0 \
    -fr_transient 5.0 \
    -fr_transient_set 2.0 \
    -unit_design_element_size_permanent 1.0 \
    -unit_design_element_size_transient 1.0 \
    -unit_design_element_size_transient_set 1.0
```

Prerequisite: Create Failure Mode (FM) - ISO 26262

```
create_fm ALU_StuckAtZero \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 0.0 \
    -safeness_transient_estimated 0.0 \
    -pvsg_permanent 95.0 \
    -pvsg_transient 90.0 \
    -perceived_primary_permanent 0.0 \
    -perceived_primary_transient 0.0 \
    -perceived_secondary_permanent 0.0 \
    -perceived_secondary_transient 0.0 \
    -no_effect_permanent {} \
    -no_effect_transient {}

create_fm Register_BitFlip \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 0.0 \
    -safeness_transient_estimated 10.0 \
    -pvsg_permanent 80.0 \
    -pvsg_transient 70.0 \
    -perceived_primary_permanent 0.0 \
    -perceived_primary_transient 0.0 \
    -perceived_secondary_permanent 0.0 \
    -perceived_secondary_transient 0.0 \
    -no_effect_permanent {} \
```

```
    -no_effect_transient {}

create_fm ALU_Overflow \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 5.0 \
    -safeness_transient_estimated 15.0 \
    -pvsg_permanent 85.0 \
    -pvsg_transient 75.0 \
    -perceived_primary_permanent 0.0 \
    -perceived_primary_transient 0.0 \
    -perceived_secondary_permanent 0.0 \
    -perceived_secondary_transient 0.0 \
    -no_effect_permanent {} \
    -no_effect_transient {}
```

Assignment Examples

Example 1: Assumption-Based FMEDA with Absolute FM Size

This example demonstrates assumption-based FMEDA where the failure mode size is specified as an absolute value in technology element units for an ISO 26262 project.

```
assign_te_fm \
    -te_name CMOS_28nm_Logic \
    -fm_name ALU_StuckAtZero \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -fm_size_type_assumption_based absolute \
    -fm_size_type_calculation_based none \
    -fm_size_permanent_assumption_based 1500.0 \
    -fm_size_transient_assumption_based 1500.0 \
    -fm_size_transient_set_assumption_based 1500.0 \
    -fm_size_permanent_calculation_based {} \
    -fm_size_transient_calculation_based {} \
    -fm_size_transient_set_calculation_based {} \
    -fm_mapping_format {} \
    -fm_mapping {} \
    -fm_mapping_exclude {} \
    -user_defined_arguments {}
```

Example 2: Calculation-Based FMEDA with Design Hierarchy Mapping

This example demonstrates calculation-based FMEDA where the failure mode size is calculated based on design hierarchy mapping using the mapping_scaling method. A scaling factor is applied to the mapped design portion.

```
assign_te_fm \
    -te_name CMOS_28nm_Logic \
    -fm_name Register_BitFlip \
    -parent_element ALU_Part \
```

```
    -parent_fmeda SoC_FMEDA \
    -fm_size_type_assumption_based none \
    -fm_size_type_calculation_based mapping_scaling \
    -fm_size_permanent_assumption_based {} \
    -fm_size_transient_assumption_based {} \
    -fm_size_transient_set_assumption_based {} \
    -fm_size_permanent_calculation_based 25.0 \
    -fm_size_transient_calculation_based 25.0 \
    -fm_size_transient_set_calculation_based 25.0 \
    -fm_mapping_format sv \
    -fm_mapping {top.cpu.alu.register_file} \
    -fm_mapping_exclude {top.cpu.alu.register_file.scan_chain} \
    -user_defined_arguments {}
```

Example 3: Assumption-Based FMEDA with Percentage FM Size

This example demonstrates assumption-based FMEDA where the failure mode size is specified as a percentage of the parent element size assigned via assign_te_element.

```
assign_te_fm \
    -te_name CMOS_28nm_Logic \
    -fm_name ALU_Overflow \
    -parent_element ALU_Part \
    -parent_fmeda SoC_FMEDA \
    -fm_size_type_assumption_based percentage \
    -fm_size_type_calculation_based none \
    -fm_size_permanent_assumption_based 20.0 \
    -fm_size_transient_assumption_based 20.0 \
    -fm_size_transient_set_assumption_based 20.0 \
    -fm_size_permanent_calculation_based {} \
    -fm_size_transient_calculation_based {} \
    -fm_size_transient_set_calculation_based {} \
    -fm_mapping_format {} \
    -fm_mapping {} \
    -fm_mapping_exclude {} \
    -user_defined_arguments {}
```

5.6.11 assign_sm_fm

Table 22—assign_sm_fm command

Purpose	Assign a safety mechanism to a failure mode

Syntax	**assign_sm_fm** **-sm_name** *sm_name* **-fm_name** *fm_name* **-parent_element** *parent_element* **-parent_fmeda** *parent_fmeda* [**-sm_parent_fmeda** *sm_parent_fmeda*] [**-dc_permanent_estimated** *float* [*0.0*, *100.0*]] [**-dc_transient_estimated** *float* [*0.0*, *100.0*]] [**-dc_latent_primary_estimated** *float* [*0.0*, *100.0*]] [**-dc_latent_secondary_estimated** *float* [*0.0*, *100.0*]] [**-dc_permanent_measured** *float* [*0.0*, *100.0*]] [**-dc_transient_measured** *float* [*0.0*, *100.0*]] [**-dc_latent_measured** *float* [*0.0*, *100.0*]] [**-dc_latent_primary_measured** *float* [*0.0*, *100.0*]] [**-dc_latent_secondary_measured** *float* [*0.0*, *100.0*]] **-active** [< **TRUE** | **FALSE** >] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-sm_name** *sm_name*	The name of the safety mechanism applied to the failure mode
	-fm_name *fm_name*	The name of the failure mode covered by the safety mechanism
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the failure mode
	-parent_fmeda *parent_fmeda*	The parent FMEDA associated with the failure mode
	-sm_parent_fmeda *sm_parent_fmeda*	The parent FMEDA associated with the safety mechanism
	-dc_permanent_estimated *float* [*0.0*, *100.0*]	The estimated diagnostic coverage of the safety mechanism for permanent faults
	-dc_transient_estimated *float* [*0.0*, *100.0*]	The estimated diagnostic coverage of the safety mechanism for transient faults
	-dc_latent_primary_estimated *float* [*0.0*, *100.0*]	The estimated diagnostic coverage of the safety mechanism for latent primary faults as defined in ISO 26262-2018, Part 11
	-dc_latent_secondary_estimated *float* [*0.0*, *100.0*]	The estimated diagnostic coverage of the safety mechanism for latent secondary faults as defined in ISO 26262-2018, Part 11
	-dc_permanent_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage of the safety mechanism for permanent faults as a result of fault injection activities
	-dc_transient_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage of the safety mechanism for transient faults as a result of fault injection activities
	-dc_latent_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage of the safety mechanism for latent faults as a result of fault injection activities
	-dc_latent_primary_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage of the safety mechanism for latent primary faults as a result of fault injection activities
	-dc_latent_secondary_measured *float* [*0.0*, *100.0*]	The measured diagnostic coverage of the safety mechanism for latent secondary faults as a result of fault injection activities
	-active [< **TRUE** | **FALSE** >]	Indicates whether the safety mechanism is enabled for this failure mode
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

assign_sm_fm command assigns a *safety mechanism* to a *failure mode* within a specified *FMEDA*.

- **-sm_name** — specifies the name of the *safety mechanism* applied to the *failure mode*.
- **-fm_name** — specifies the name of the *failure mode* covered by the *safety mechanism*.
- **-parent_element** — specifies the identifier of the parent *element* to which this *failure mode* is connected in the *FS hierarchy*, enabling unique identification of the *failure mode*.
- **-parent_fmeda** — specifies the identifier of the parent *FMEDA* to which the *failure mode* belongs.
- **-sm_parent_fmeda** — specifies the identifier of the parent *FMEDA* in which the *safety mechanism* is defined.
- **-dc_permanent_estimated**, **-dc_transient_estimated**, **-dc_latent_primary_estimated**, and **-dc_latent_secondary_estimated** — specify the estimated diagnostic coverage of the *safety mechanism* for permanent, transient, latent primary, and latent secondary faults, respectively, as defined in ISO 26262-2018, Part 11. Values are given as percentages in the range [0.0, 100.0].
- **-dc_permanent_measured**, **-dc_transient_measured**, **-dc_latent_measured**, **-dc_latent_primary_measured**, and **-dc_latent_secondary_measured** — specify the measured diagnostic coverage of the *safety mechanism* for permanent, transient, latent, latent primary, and latent secondary faults, respectively, as a result of fault injection or verification activities. Values are given as percentages in the range [0.0, 100.0].
- **-active** — indicates whether the *safety mechanism* is enabled for this *failure mode*. **assign_sm_fm -active** can only be specified as TRUE or FALSE if **create_sm -configurable** is set to TRUE.

Rules

The following rules apply:

Table 23—Rules for ‘assign_sm_fm’ command

Rule ID	Rule
Rule_SM_FM_1	It shall be an error if **-active** is specified as TRUE or FALSE in **assign_sm_fm** and **-configurable** is not defined as TRUE in the corresponding **create_sm** command.
Rule_SM_FM_2	The **-dc_*** values in **assign_sm_fm** are not required and, if not specified, shall be inherited from the corresponding **-dc_*** values of the related **create_sm** command.
Rule_SM_FM_3	If **-dc_*** values are specified in both **create_sm** and **assign_sm_fm**, the value defined in **assign_sm_fm** shall take precedence.

How-to Examples

Prerequisite Examples

These examples show the prerequisite commands needed before using assign_sm_fm.

Prerequisite: Create SM

```
create_sm Parity_Check \
    -configurable FALSE \
    -dc_permanent 95.0 \
    -dc_transient 90.0 \
    -dc_latent_primary 85.0 \
    -dc_latent_secondary 80.0

create_sm Watchdog_Timer \
    -configurable TRUE \
    -dc_permanent 85.0 \
```

```
    -dc_transient 88.0 \
    -dc_latent_primary 75.0 \
    -dc_latent_secondary 70.0
```

Prerequisite: Create FM

```
create_fm ALU_Computation_Error \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 5.0 \
    -safeness_transient_estimated 10.0 \
    -pvsg_permanent 85.0 \
    -pvsg_transient 80.0 \
    -perceived_primary_permanent 10.0 \
    -perceived_primary_transient 8.0 \
    -perceived_secondary_permanent 2.0 \
    -perceived_secondary_transient 1.0 \
    -no_effect_permanent {} \
    -no_effect_transient {}
```

Assignment Examples

Example 1: Assignment of Configurable Safety Mechanism (Active)

This example assigns a configurable watchdog timer safety mechanism to a failure mode. The SM is enabled (-active TRUE) and inherits the default DC values from create_sm per Rule_SM_FM_2. Per Rule_SM_FM_1, -active can only be specified when -configurable TRUE in create_sm.

```
assign_sm_fm \
    -sm_name Watchdog_Timer \
    -fm_name ALU_Computation_Error \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -active TRUE \
    -user_defined_arguments {}
```

Example 2: Assignment with Diagnostic Coverage Overrides

This example assigns a configurable safety mechanism with FM-specific DC values. Per Rule_SM_FM_3, the DC values specified here override those from create_sm. The estimated DC values for permanent and transient faults are provided for this specific FM.

```
assign_sm_fm \
    -sm_name Watchdog_Timer \
    -fm_name ALU_Computation_Error \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -dc_permanent_estimated 92.0 \
    -dc_transient_estimated 90.0 \
    -active TRUE \
```

```
    -user_defined_arguments {}
```

Example 3: Assignment of Non-Configurable Safety Mechanism

This example assigns a non-configurable parity check safety mechanism to a failure mode. Since -configurable FALSE in create_sm, the -active argument cannot be used per Rule_SM_FM_1. DC values are inherited from create_sm per Rule_SM_FM_2.

```
assign_sm_fm \
    -sm_name Parity_Check \
    -fm_name ALU_Computation_Error \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -user_defined_arguments {}
```

5.6.12 assign_fm_fme

Table 24—assign_fm_fme command

Purpose	Assign a failure mode to one or more failure mode effects	
Syntax	**assign_fm_fme** **-fm_name** *fm_name* **-parent_element** *parent_element* **-fme_list** *fme_list_list* **-parent_fmeda** *parent_fmeda* **-fme_weights** *list_of_floats* [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-fm_name** *fm_name*	The name of the failure mode contributing to the failure mode effect
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the failure mode
	-fme_list *fme_list_list*	The list of failure mode effects caused by the failure mode. Each failure mode effect represents the top-level consequence within the defined scope
	-parent_fmeda *parent_fmeda*	The FMEDA associated with the assignment
	-fme_weights *list_of_floats*	The weights of the contributions of the failure mode to the failure mode effects defined in -fme_list
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

assign_fm_fme command links a single *failure mode* identified by **-fm_name** to one or more *failure mode effects*, i.e., entries in **-fme_list**.

The relationship between *failure mode* and *failure mode effect* is many-to-many:

- A *failure mode* can contribute to multiple *failure mode effects*.
- A *failure mode effect* can be caused by multiple *failure modes*.

The assignment of *failure modes* to *failure mode effects* is not restricted by the position of the *failure mode* within the *FS hierarchy*. The total failure rate for the device due to all *failure modes* is conserved when

distributed across the *failure mode effects*; accordingly, all *failure modes* shall be associated with at least one *failure mode effect*.

- **-parent_element** — specifies the identifier of the parent *element* to which the *failure mode* is connected in the *FS hierarchy*, enabling unique identification of the *failure mode*.
- **-fme_list** — enumerates the assigned *failure mode effects*, while **-fme_weights** provides the percentage contribution [0.0, 100.0] of the *failure mode* to each effect in **-fme_list**; the number of entries shall match the list length of **-fme_list**, all entries w_k shall be non-negative, and the sum of all weights for a given *failure mode* across all **assign_fm_fme** commands shall equal 100 (see Rule_FM_FME_1):

$$\sum_{k=1}^{N} w_k = 100, \quad w_k \geq 0 \tag{2}$$

- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *failure mode* and *failure mode effect* belong.

Rules

The following rules apply:

Table 25—Rules for 'assign_fm_fme' command

Rule ID	Rule
Rule_FM_FME_1	It shall be an error if the sum of the values specified in **-fme_weights** for a given *failure mode* does not equal 100%; this assumes that the *failure mode effects* are non-overlapping.

How-to Examples

Prerequisite Examples

These examples show the prerequisite commands needed before using assign_fm_fme.

Prerequisite: Create Failure Modes

```
create_fm ALU_Stuck_At_Fault \
    -parent_element CPU_Core \
    -parent_fmeda Automotive_ECU_FMEDA \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 5.0 \
    -safeness_transient_estimated 10.0 \
    -pvsg_permanent 85.0 \
    -pvsg_transient 80.0 \
    -perceived_primary_permanent 8.0 \
    -perceived_primary_transient 5.0 \
    -perceived_secondary_permanent 2.0 \
    -perceived_secondary_transient 1.0 \
    -no_effect_permanent {} \
    -no_effect_transient {} \
    -user_defined_arguments {}
```

```
create_fm Memory_Bit_Flip \
    -parent_element SRAM_Block \
    -parent_fmeda Automotive_ECU_FMEDA \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 2.0 \
    -safeness_transient_estimated 15.0 \
    -pvsg_permanent 90.0 \
    -pvsg_transient 75.0 \
    -perceived_primary_permanent 10.0 \
    -perceived_primary_transient 8.0 \
    -perceived_secondary_permanent 3.0 \
    -perceived_secondary_transient 2.0 \
    -no_effect_permanent {} \
    -no_effect_transient {} \
    -user_defined_arguments {}

create_fm Clock_Glitch \
    -parent_element Clock_Gen \
    -parent_fmeda Industrial_PLC_FMEDA \
    -safety_relevant {} \
    -no_part TRUE \
    -safeness_permanent_estimated 5.0 \
    -safeness_transient_estimated 10.0 \
    -pvsg_permanent {} \
    -pvsg_transient {} \
    -perceived_primary_permanent {} \
    -perceived_primary_transient {} \
    -perceived_secondary_permanent {} \
    -perceived_secondary_transient {} \
    -no_effect_permanent 10.0 \
    -no_effect_transient 20.0 \
    -user_defined_arguments {}
```

Prerequisite: Create Failure Mode Effects

```
create_fme Loss_of_Vehicle_Control \
    -parent_fmeda Automotive_ECU_FMEDA \
    -user_defined_arguments {}

create_fme Erroneous_Sensor_Reading \
    -parent_fmeda Automotive_ECU_FMEDA \
    -user_defined_arguments {}

create_fme Data_Corruption \
    -parent_fmeda Automotive_ECU_FMEDA \
    -user_defined_arguments {}

create_fme Process_Shutdown \
    -parent_fmeda Industrial_PLC_FMEDA \
    -user_defined_arguments {}

create_fme Output_Delay \
```

```
    -parent_fmeda Industrial_PLC_FMEDA \
    -user_defined_arguments {}
```

Assignment Examples

Example 1: Single Failure Mode to Single Effect (100% Weight)

This example assigns an ALU fault that contributes 100% to loss of vehicle control, demonstrating a direct one-to-one mapping per Rule_FM_FME_1.

```
assign_fm_fme \
    -fm_name ALU_Stuck_At_Fault \
    -parent_element CPU_Core \
    -fme_list {Loss_of_Vehicle_Control} \
    -parent_fmeda Automotive_ECU_FMEDA \
    -fme_weights {100} \
    -user_defined_arguments {}
```

Example 2: Single Failure Mode to Multiple Effects with Proportional Weights

This example assigns a memory bit flip to multiple effects with weights summing to 100, demonstrating Rule_FM_FME_1 compliance and many-to-many relationship.

```
assign_fm_fme \
    -fm_name Memory_Bit_Flip \
    -parent_element SRAM_Block \
    -fme_list {Data_Corruption Erroneous_Sensor_Reading} \
    -parent_fmeda Automotive_ECU_FMEDA \
    -fme_weights {70 30} \
    -user_defined_arguments {}
```

Example 3: Cross-Hierarchy Assignment (IEC 61508)

This example assigns a clock glitch from Clock_Gen element to multiple effects in an industrial PLC context, demonstrating cross-hierarchy assignment and IEC 61508 usage.

```
assign_fm_fme \
    -fm_name Clock_Glitch \
    -parent_element Clock_Gen \
    -fme_list {Process_Shutdown Output_Delay} \
    -parent_fmeda Industrial_PLC_FMEDA \
    -fme_weights {60 40} \
    -user_defined_arguments {}
```

5.6.13 assign_fm_sreq

Table 26—assign_fm_sreq command

Purpose	Assign a failure mode to a safety requirement

Syntax	**assign_fm_sreq** **-fm_name** *fm_name* **-sreq_name** *sreq_name* **-parent_element** *parent_element* **-parent_fmeda** *parent_fmeda* **-safety_relevant** [< **TRUE** | **FALSE** >] **-no_part** [< **TRUE** | **FALSE** >] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-fm_name** *fm_name*	The name of the failure mode
	-sreq_name *sreq_name*	The name of the safety requirement
	-parent_element *parent_element*	The parent element in the FS hierarchy associated with the failure mode
	-parent_fmeda *parent_fmeda*	The FMEDA associated with the assignment
	-safety_relevant [< **TRUE** | **FALSE** >]	Indicates whether the failure mode is safety-relevant in the context of ISO 26262 for the given safety requirement
	-no_part [< **TRUE** | **FALSE** >]	Indicates whether the failure mode is classified as having no part in a dangerous failure in the context of IEC 61508 for the given safety requirement. When set to TRUE, the failure mode does not contribute to a dangerous condition for the specified safety requirement
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return an empty string if successful, or raise a TCL_ERROR if not.	

assign_fm_sreq command assigns a *failure mode* to a *safety requirement* within a specified *FMEDA*.

The parameters of a *failure mode* are defined by the **create_fm** command, while the **assign_fm_sreq** command specifies how those parameters are applied in the metrics calculation for each *safety requirement*.

The link between a *failure mode effect* and a *safety goal* shall be implicitly created by the mapping of *failure mode* to *failure mode effect* (FM-to-FME) and *failure mode* to *safety goal* (FM-to-SG).
The *failure mode effect* metrics for a given *safety goal* shall be calculated considering how the *failure mode*, assigned to that *failure mode effect* by weights, has been classified for that *safety goal*.

- **-fm_name** — specifies the identifier of the *failure mode* to be assigned.
- **-sreq_name** — specifies the name of the *safety requirement* to which the *failure mode* is mapped.
- **-parent_element** — specifies the identifier of the parent *element* to which the *failure mode* is connected in the *FS hierarchy*, enabling unique identification of the *failure mode*.
- **-parent_fmeda** — specifies the name of the *FMEDA* to which the *failure mode* and *safety requirement* belong.
- **-safety_relevant** — indicates whether the *failure mode* is safety-relevant for the specified *safety requirement* in the context of ISO 26262.
- **-no_part** — indicates whether the *failure mode* is classified as having no part in a dangerous failure for the specified *safety requirement* in the context of IEC 61508. When set to TRUE, the *failure mode* does not contribute to a dangerous condition for the specified *safety requirement*.

How-to Examples

Prerequisite Examples

These examples show the prerequisite commands needed before using assign_fm_sreq.

Prerequisite: Create Failure Modes

```
create_fm ALU_Computation_Error \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -safety_relevant TRUE \
    -no_part {} \
    -safeness_permanent_estimated 5.0 \
    -safeness_transient_estimated 10.0 \
    -pvsg_permanent 85.0 \
    -pvsg_transient 80.0 \
    -perceived_primary_permanent 10.0 \
    -perceived_primary_transient 8.0 \
    -perceived_secondary_permanent 2.0 \
    -perceived_secondary_transient 1.0 \
    -no_effect_permanent {} \
    -no_effect_transient {} \
    -user_defined_arguments {}

create_fm Memory_Data_Corruption \
    -parent_element Memory_Controller \
    -parent_fmeda Memory_Safety_Analysis \
    -safety_relevant {} \
    -no_part TRUE \
    -safeness_permanent_estimated 8.0 \
    -safeness_transient_estimated 12.0 \
    -pvsg_permanent {} \
    -pvsg_transient {} \
    -perceived_primary_permanent {} \
    -perceived_primary_transient {} \
    -perceived_secondary_permanent {} \
    -perceived_secondary_transient {} \
    -no_effect_permanent 15.0 \
    -no_effect_transient 18.0 \
    -user_defined_arguments {}
```

Prerequisite: Create Safety Requirements

```
create_sreq Computation_Integrity_SG \
    -sreq_type safety_goal \
    -user_defined_arguments {}

create_sreq Memory_Integrity_TLSR \
    -sreq_type top_level_safety_requirement \
    -user_defined_arguments {}
```

Assignment Examples

Example 1: Basic Assignment of FM to Safety Goal (ISO 26262)

This example assigns a failure mode to a safety goal in an ISO 26262 FMEDA. The failure mode is marked as safety relevant for this safety requirement.

```
assign_fm_sreq \
    -fm_name ALU_Computation_Error \
    -sreq_name Computation_Integrity_SG \
    -parent_element CPU_Core \
    -parent_fmeda CPU_Safety_Analysis \
    -safety_relevant TRUE \
    -no_part {} \
    -user_defined_arguments {}
```

Example 2: Assignment for IEC 61508 with No-Part Classification

This example assigns a failure mode to a top-level safety requirement in an IEC 61508 FMEDA. The failure mode is marked as no_part TRUE, indicating it has no part in a dangerous failure as defined in IEC 61508 — i.e., it does not contribute to a dangerous condition for the specified safety requirement.

```
assign_fm_sreq \
    -fm_name Memory_Data_Corruption \
    -sreq_name Memory_Integrity_TLSR \
    -parent_element Memory_Controller \
    -parent_fmeda Memory_Safety_Analysis \
    -safety_relevant {} \
    -no_part TRUE \
    -user_defined_arguments {}
```

5.6.14 define_fr_iso26262

Table 27—define_fr_iso26262 command

Purpose	Return a failure rate value for a failure rate type defined in ISO 26262
Syntax	**define_fr_iso26262** **-fr_type** <**intrinsic** | **sr** | **nsr** | **safe** | **non_safe** | **spf** | **residual** | **mpf** | **mpf_detected** | **mpf_perceived_latent** | **mpf_perceived** | **mpf_latent** | **mpf_primary** | **mpf_detected_primary** | **mpf_perceived_latent_primary** | **mpf_perceived_primary** | **mpf_latent_primary** | **mpf_secondary** | **mpf_detected_secondary** | **mpf_perceived_latent_secondary** | **mpf_perceived_secondary** | **mpf_latent_secondary** > **-dc_type** <**estimated** | **measured** > **-fmeda_type** <**assumption-based** | **calculation-based** > **-scope** <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** > **-te_name** *te_name* **-analysis_type** <**permanent** | **transient** | **permanent+transient** > **-parent_fmeda** *parent_fmeda* [**-fmeda_name** *fmeda_name*] **-element_name** *element_name* **-failure_mode_name** *failure_mode_name* **-failure_mode_effect_name** *failure_mode_effect_name* **-parent_element** *parent_element* [**-sreq_name** *sreq_name_list*] **-fr_value** *float* [*0.0*, *n*] [**-user_defined_arguments** *list_of_tuples*]
Arguments	**-fr_type** <**intrinsic** | **sr** | **nsr** | **safe** | **non_safe** | **spf** | **residual** | **mpf** | **mpf_detected** | **mpf_perceived_latent** | **mpf_perceived** | **mpf_latent** | **mpf_primary** | **mpf_detected_primary** | **mpf_perceived_latent_primary** | **mpf_perceived_primary** | **mpf_latent_primary** | **mpf_secondary** | **mpf_detected_secondary** | **mpf_perceived_latent_secondary** | **mpf_perceived_secondary** | **mpf_latent_secondary** > — The failure rate type calculated according to ISO 26262

	-dc_type <**estimated** | **measured** >	The diagnostic coverage type used for reporting the estimated or measured failure rate
	-fmeda_type <**assumption-based** | **calculation-based** >	The type of FMEDA used to calculate the size of elements and failure modes
	-scope <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** >	The scope for which the failure rate is calculated
	-te_name *te_name*	The technology element for which the failure rate is calculated
	-analysis_type <**permanent** | **transient** | **permanent+transient** >	The analysis type for which the failure rate is calculated
	-parent_fmeda *parent_fmeda*	The FMEDA within which the failure rate is calculated; identifies the active FMEDA context for the operation
	-fmeda_name *fmeda_name*	When **-scope** is **fmeda** and the *FMEDA* is hierarchical, the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the failure rate is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*
	-element_name *element_name*	The name of the element in the FS hierarchy. *This argument is required only if -scope is element
	-failure_mode_name *failure_mode_name*	The name of the failure mode. *This argument is required only if -scope is failure_mode
	-failure_mode_effect_name *failure_mode_effect_name*	The name of the failure mode effect. *This argument is required only if -scope is failure_mode_effect
	-parent_element *parent_element*	The parent element in the FS hierarchy
	-sreq_name *sreq_name_list*	The safety requirement(s) for which the failure rate is calculated
	-fr_value *float* [*0.0*, *n*]	The value of the failure rate
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return the computed failure rate value as a float if successful, or raise a TCL_ERROR if not.	

define_fr_iso26262 command enables calculation and reporting of ISO 26262 failure rates at various levels of the *FS hierarchy*, supporting both estimated and measured diagnostic coverage, and both assumption-based and calculation-based FMEDA methodologies.

define_fr_iso26262 command returns a failure rate value for a specified failure rate type as defined in ISO 26262.

- **-fr_type** — specifies the failure rate type to be calculated according to ISO 26262. Supported values include intrinsic, sr, nsr, safe, non_safe, spf, residual, mpf, mpf_detected, mpf_perceived_latent, mpf_perceived, mpf_latent, mpf_primary, mpf_detected_primary, mpf_perceived_latent_primary, mpf_perceived_primary, mpf_latent_primary, mpf_secondary, mpf_detected_secondary, mpf_perceived_latent_secondary, mpf_perceived_secondary, and mpf_latent_secondary.
- **-dc_type** — specifies whether the diagnostic coverage used for the failure rate calculation is estimated or measured.
- **-te_name** — specifies the *technology element* for which the failure rate is calculated.
- **-fmeda_type** — specifies the FMEDA methodology used to determine the size of *elements* and *failure modes*. Supported values are assumption-based and calculation-based. For assumption-based, the user-assigned values take precedence; for calculation-based, the design mapping based values take precedence.
- **-analysis_type** — specifies the analysis type for which the failure rate is calculated. Supported values are permanent, transient, and permanent+transient, which sums over the failure rates for both permanent and transient faults.
- **-parent_fmeda** — specifies the *FMEDA* within which the failure rate is calculated, identifying the active *FMEDA* context for the operation.
- **-fmeda_name** — when **-scope** is `fmeda` and the *FMEDA* is hierarchical, specifies the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the failure rate is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*.
- **-scope** — specifies the scope for which the failure rate is calculated. Supported values are fmeda, element, failure_mode, and failure_mode_effect.
- **-element_name** — specifies the name of the *element* in the *FS hierarchy*. This field is mandatory if **-scope** is element.
- **-failure_mode_name** — specifies the name of the *failure mode*. This field is mandatory if **-scope** is failure_mode.
- **-failure_mode_effect_name** — specifies the name of the *failure mode effect*. This field is mandatory if **-scope** is failure_mode_effect.
- **-parent_element** — specifies the identifier of the parent *element* to which the scope (*element*/*failure mode*) is connected in the *FS hierarchy*, enabling unique identification of the scope (*element*/*failure mode*). This field is mandatory if **-scope** is element or failure_mode.
- **-sreq_name** — specifies the list of *safety requirements* for which the failure rate is calculated. If not specified, the *failure mode* to *safety requirement* assignment is not considered in the failure rate calculation.
- **-fr_value** — specifies the value of the failure rate [unit: FIT for Failure in Time].

Rules

The following rules apply:

Table 28—Rules for 'define_fr_iso26262' command

Rule ID	Rule
Rule_FR_ISO_1	When **-dc_type** is set to estimated, ISO 26262 failure rates are calculated based on diagnostic coverage defined by one of the following: - **-dc** specified in **create_sm** (isolated DC) - **-dc_*_estimated** assigned by **assign_sm_fm** (assigned estimated DC) - **-dc_*_expert** specified in **create_fm** (expert DC) The following priority rules apply: - When both isolated DC and assigned estimated DC exist for the same object, assigned estimated DC shall take precedence over isolated DC. - When expert DC exists for the same object, expert DC shall take precedence over assigned estimated DC or isolated DC.NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_FR_ISO_2	When **-dc_type** is set to measured, ISO 26262 failure rates are calculated based on diagnostic coverage defined by one of the following: - **-dc_*_measured** assigned by **assign_sm_fm** (assigned measured DC) - **-dc_*_measured** specified in **create_fm** (aggregated over the *safety mechanisms* associated with the *failure mode*)When **-dc_measured** is not available, the command shall fall back to either isolated DC, assigned estimated DC, or expert DC, as defined in Rule_FR_ISO_1. NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_FR_ISO_3	When both user-defined area estimations (FM_size_*_assumption_based) and design mapping exist for the same *failure mode*, the following shall apply for ISO 26262 failure rates calculation: - If **-fmeda_type** is set to assumption-based: FM_size_*_assumption_based shall take precedence over FM_size_*_calculation_based. - If **-fmeda_type** is set to calculation-based: FM_size_*_calculation_based shall take precedence over FM_size_*_assumption_based.When only one of the two sources is available, the available value shall apply.
Rule_FR_ISO_4	When ISO 26262 failure rates are reported with **-sreq_name** specified, the following shall apply: A *failure mode* shall be considered safety relevant if it is classified as safety relevant for at least one of the specified *safety requirements*.
Rule_FR_ISO_5	When ISO 26262 failure rates are reported without specifying **-sreq_name**, the following shall apply: Failure rates shall be reported independently of *safety requirements*.

How-to Examples

Overview

The define_fr_iso26262 command calculates and returns ISO 26262 failure rates at various levels of the functional safety hierarchy. It supports both estimated and measured diagnostic coverage, assumption-based and calculation-based FMEDA methodologies, and various analysis scopes.

Example 1: FMEDA-Level Single Point Fault Calculation for Automotive MCU

This example calculates the SPF failure rate at the FMEDA level for an automotive microcontroller SoC using estimated diagnostic coverage and assumption-based FMEDA methodology for safety-critical applications.

```
define_fr_iso26262 \
```

```
    -fr_type spf \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_16nm \
    -analysis_type permanent \
    -parent_fmeda MCU_FMEDA \
    -fr_value 125.7 \
    -user_defined_arguments {}
```

Example 2: Element-Level Residual Failure Rate for CPU IP Core

This example calculates the residual failure rate for a CPU core IP element within an automotive SoC using measured diagnostic coverage from RTL simulation and calculation-based FMEDA for silicon verification.

```
define_fr_iso26262 \
    -fr_type residual \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope element \
    -te_name CMOS_16nm \
    -analysis_type permanent \
    -parent_fmeda MCU_FMEDA \
    -element_name CPU_Core \
    -parent_element SoC_Top \
    -fr_value 8.3 \
    -user_defined_arguments {}
```

Example 3: Combined Fault Analysis for Automotive Actuator SoC

This example calculates MPF detected secondary failure rate combining both permanent and transient faults at the FMEDA level for an actuator control SoC using measured diagnostic coverage from fault injection campaigns.

```
define_fr_iso26262 \
    -fr_type mpf_detected_secondary \
    -dc_type measured \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_7nm \
    -analysis_type permanent+transient \
    -parent_fmeda Actuator_FMEDA \
    -fr_value 18.9 \
    -user_defined_arguments {}
```

Example 4: Safety-Related Failure Rate for Brake ECU Controller IP

This example calculates safety-related failure rate for a brake controller IP block within an automotive safety ECU SoC, filtering by multiple ASIL-D safety requirements per Rule_FR_ISO_4 for brake-by-wire systems.

```
define_fr_iso26262 \
```

```
    -fr_type sr \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope element \
    -te_name CMOS_22nm \
    -analysis_type permanent \
    -parent_fmeda Brake_ECU_FMEDA \
    -element_name Brake_Controller \
    -parent_element Brake_ECU \
    -sreq_name {SREQ_BR_001 SREQ_BR_002 SREQ_BR_005} \
    -fr_value 89.3 \
    -user_defined_arguments {}
```

Example 5: Failure Mode Level NSR Calculation for Memory Controller IP

This example calculates the non-safety-related failure rate for a stuck-at-1 failure mode in a DDR memory controller IP within an automotive infotainment SoC, demonstrating failure mode level analysis with estimated diagnostic coverage for ASIL-B systems.

```
define_fr_iso26262 \
    -fr_type nsr \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope failure_mode \
    -te_name CMOS_16nm \
    -analysis_type permanent \
    -parent_fmeda Infotainment_FMEDA \
    -element_name DDR_Controller \
    -failure_mode_name Stuck_At_1 \
    -parent_element Infotainment_SoC \
    -fr_value 12.6 \
    -user_defined_arguments {}
```

Example 6: Granular Failure Mode Effect Analysis for Sensor IP

This example demonstrates failure mode effect level calculation for a torque sensor IP within an automotive steering SoC, analyzing gradual drift failure effects for ASIL-C electric power steering systems with comprehensive fault coverage.

```
define_fr_iso26262 \
    -fr_type mpf_perceived_latent \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope failure_mode_effect \
    -te_name CMOS_16nm \
    -analysis_type permanent+transient \
    -parent_fmeda Steering_FMEDA \
    -failure_mode_effect_name Gradual_Offset \
    -sreq_name {SREQ_ST_010 SREQ_ST_015} \
    -fr_value 3.72 \
    -user_defined_arguments {}
```

5.6.15 define_fr_iec61508

Table 29—define_fr_iec61508 command

Purpose	Return a failure rate value for a failure rate type defined in IEC 61508	
Syntax	**define_fr_iec61508** **-fr_type** <**total** | **safe** | **safe_detected** | **safe_undetected** | **dangerous** | **dangerous_detected** | **dangerous_undetected** | **no_effect** > **-dc_type** <**estimated** | **measured** > **-fmeda_type** <**assumption-based** | **calculation-based** > **-scope** <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** > **-te_name** *te_name* **-analysis_type** <**permanent** | **transient** | **permanent+transient** > **-parent_fmeda** *parent_fmeda* [**-fmeda_name** *fmeda_name*] **-element_name** *element_name* **-failure_mode_name** *failure_mode_name* **-failure_mode_effect_name** *failure_mode_effect_name* **-parent_element** *parent_element* [**-sreq_name** *sreq_name_list*] **-fr_value** *float* [*0.0*, *n*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-fr_type** <**total** | **safe** | **safe_detected** | **safe_undetected** | **dangerous** | **dangerous_detected** | **dangerous_undetected** | **no_effect** >	The failure rate type calculated according to IEC 61508. Values include total (λ), safe (λ_S), safe_detected (λ_{SD}), safe_undetected (λ_{SU}), dangerous (λ_D), dangerous_detected (λ_{DD}), dangerous_undetected (λ_{DU}), and no_effect (λ_{NE}) as defined in IEC 61508
	-dc_type <**estimated** | **measured** >	The diagnostic coverage type used for reporting the estimated or measured failure rate
	-fmeda_type <**assumption-based** | **calculation-based** >	The type of FMEDA used to calculate the size of elements and failure modes
	-scope <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** >	The scope for which the failure rate is calculated
	-te_name *te_name*	The technology element for which the failure rate is calculated
	-analysis_type <**permanent** | **transient** | **permanent+transient** >	The analysis type for which the failure rate is calculated
	-parent_fmeda *parent_fmeda*	The FMEDA within which the failure rate is calculated; identifies the active FMEDA context for the operation
	-fmeda_name *fmeda_name*	When **-scope** is **fmeda** and the *FMEDA* is hierarchical, the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the failure rate is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*
	-element_name *element_name*	The name of the element in the FS hierarchy. *This argument is required only if -scope is element
	-failure_mode_name *failure_mode_name*	The name of the failure mode. *This argument is required only if -scope is failure_mode
	-failure_mode_effect_name *failure_mode_effect_name*	The name of the failure mode effect. *This argument is required only if -scope is failure_mode_effect
	-parent_element *parent_element*	The parent element in the FS hierarchy

	-sreq_name *sreq_name_list*	The safety requirement(s) for which the failure rate is calculated
	-fr_value *float* [*0.0*, *n*]	The value of the failure rate
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return the computed failure rate value as a float if successful, or raise a TCL_ERROR if not.	

define_fr_iec61508 command enables calculation and reporting of IEC 61508 failure rates at various levels of the *FS hierarchy*, supporting both estimated and measured diagnostic coverage, and both assumption-based and calculation-based FMEDA methodologies.

define_fr_iec61508 command returns a failure rate value for a specified failure rate type as defined in IEC 61508.

- **-fr_type** — specifies the failure rate type to be calculated according to IEC 61508. Supported values include total, safe, safe_detected, safe_undetected, dangerous, dangerous_detected, dangerous_undetected, and no_effect, corresponding to the failure rate categories defined in IEC 61508.
- **-dc_type** — specifies whether the diagnostic coverage used for the failure rate calculation is estimated or measured.
- **-te_name** — specifies the *technology element* for which the failure rate is calculated.
- **-fmeda_type** — specifies the FMEDA methodology used to determine the size of *elements* and *failure modes*. Supported values are assumption-based and calculation-based. For assumption-based, the user-assigned values take precedence; for calculation-based, the design mapping based values take precedence.
- **-analysis_type** — specifies the analysis type for which the failure rate is calculated. Supported values are permanent, transient, and permanent+transient, which sums over the failure rates for both permanent and transient faults.
- **-parent_fmeda** — specifies the *FMEDA* within which the failure rate is calculated, identifying the active *FMEDA* context for the operation.
- **-fmeda_name** — when **-scope** is `fmeda` and the *FMEDA* is hierarchical, specifies the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the failure rate is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*.
- **-scope** — specifies the scope for which the failure rate is calculated. Supported values are fmeda, element, failure_mode, and failure_mode_effect.
- **-element_name** — specifies the name of the *element* in the *FS hierarchy*. This field is mandatory if **-scope** is element.
- **-failure_mode_name** — specifies the name of the *failure mode*. This field is mandatory if **-scope** is failure_mode.
- **-failure_mode_effect_name** — specifies the name of the *failure mode effect*. This field is mandatory if **-scope** is failure_mode_effect.
- **-parent_element** — specifies the identifier of the parent *element* to which the scope (*element*/*failure mode*) is connected in the *FS hierarchy*, enabling unique identification of the scope (*element*/*failure mode*). This field is mandatory if **-scope** is element or failure_mode.
- **-sreq_name** — specifies the list of *safety requirements* for which the failure rate is calculated. If not specified, the *failure mode* to *safety requirement* assignment is not considered in the failure rate calculation.
- **-fr_value** — specifies the value of the failure rate [unit: FIT for Failure in Time].

Rules

The following rules apply:

Table 30—Rules for ‘define_fr_iec61508’ command

Rule ID	Rule
Rule_FR_IEC_1	When **-dc_type** is set to estimated, IEC 61508 failure rates are calculated based on diagnostic coverage defined by one of the following: - **-dc** specified in **create_sm** (isolated DC) - **-dc_*_estimated** assigned by **assign_sm_fm** (assigned estimated DC) - **-dc_*_expert** specified in **create_fm** (expert DC) The following priority rules apply: - When both isolated DC and assigned estimated DC exist for the same object, assigned estimated DC shall take precedence over isolated DC. - When expert DC exists for the same object, expert DC shall take precedence over assigned estimated DC or isolated DC.NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_FR_IEC_2	When **-dc_type** is set to measured, IEC 61508 failure rates are calculated based on diagnostic coverage defined by one of the following: - **-dc_*_measured** assigned by **assign_sm_fm** (assigned measured DC) - **-dc_*_measured** specified in **create_fm** (aggregated over the *safety mechanisms* associated with the *failure mode*)When **-dc_measured** is not available, the command shall fall back to either isolated DC, assigned estimated DC, or expert DC, as defined in Rule_FR_IEC_1. NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_FR_IEC_3	When both user-defined area estimations (FM_size_*_assumption_based) and design mapping exist for the same *failure mode*, the following shall apply for IEC 61508 failure rates calculation: - If **-fmeda_type** is set to assumption-based: FM_size_*_assumption_based shall take precedence over FM_size_*_calculation_based. - If **-fmeda_type** is set to calculation-based: FM_size_*_calculation_based shall take precedence over FM_size_*_assumption_based.When only one of the two sources is available, the available value shall apply.
Rule_FR_IEC_4	When IEC 61508 failure rates are reported with **-sreq_name** specified, the following shall apply: A *failure mode* shall be considered safety relevant if it is classified as safety relevant for at least one of the specified *safety requirements*.
Rule_FR_IEC_5	When IEC 61508 failure rates are reported without specifying **-sreq_name**, the following shall apply: Failure rates shall be reported independently of *safety requirements*.

How-to Examples

Overview

The define_fr_iec61508 command calculates and returns IEC 61508 failure rates at various levels of the functional safety hierarchy. It supports both estimated and measured diagnostic coverage, assumption-based and calculation-based FMEDA methodologies, and various analysis scopes for industrial safety-critical semiconductor systems.

Example 1: FMEDA-Level Dangerous Failure Rate for Industrial Motor Controller SoC

This example calculates the dangerous failure rate at the FMEDA level for an industrial motor controller SoC using estimated diagnostic coverage and assumption-based FMEDA methodology for SIL-rated safety-critical industrial automation.

```
define_fr_iec61508 \
    -fr_type dangerous \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_28nm \
    -analysis_type permanent \
    -parent_fmeda Motor_Controller_FMEDA \
    -fr_value 215.4 \
    -user_defined_arguments {}
```

Example 2: Element-Level Dangerous Detected Failure Rate for Process Control IP

This example calculates the dangerous detected failure rate for a PLC processor IP element within an industrial process control SoC using measured diagnostic coverage from FPGA-based fault injection and calculation-based FMEDA for silicon verification.

```
define_fr_iec61508 \
    -fr_type dangerous_detected \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope element \
    -te_name CMOS_28nm \
    -analysis_type permanent \
    -parent_fmeda PLC_FMEDA \
    -element_name PLC_Processor \
    -parent_element PLC_SoC \
    -fr_value 45.8 \
    -user_defined_arguments {}
```

Example 3: Combined Fault Analysis for Safety Relay SoC

This example calculates dangerous undetected failure rate combining both permanent and transient faults at the FMEDA level for a safety relay control SoC used in emergency shutdown systems, using measured diagnostic coverage from comprehensive fault campaigns.

```
define_fr_iec61508 \
    -fr_type dangerous_undetected \
    -dc_type measured \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_40nm \
    -analysis_type permanent+transient \
    -parent_fmeda Safety_Relay_FMEDA \
    -fr_value 12.3 \
    -user_defined_arguments {}
```

Example 4: Failure Mode Level Analysis for Power Management IP

This example calculates dangerous failure rate at the failure mode level for a power management IP within an industrial power supply SoC, analyzing voltage regulator failure modes for SIL-2 power distribution systems.

```
define_fr_iec61508 \
    -fr_type dangerous \
    -dc_type estimated \
    -fmeda_type calculation-based \
    -scope failure_mode \
    -te_name CMOS_65nm \
    -analysis_type permanent \
    -parent_fmeda Power_Supply_FMEDA \
    -element_name Voltage_Regulator \
    -failure_mode_name Overvoltage_Fault \
    -parent_element Power_Supply_SoC \
    -fr_value 28.7 \
    -user_defined_arguments {}
```

Example 5: Granular Failure Mode Effect Analysis for Sensor Interface IP

This example demonstrates failure mode effect level calculation for a temperature sensor interface IP within an industrial monitoring SoC, analyzing signal drift effects for SIL-3 process monitoring systems with comprehensive diagnostic coverage and safety requirement filtering.

```
define_fr_iec61508 \
    -fr_type dangerous_undetected \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope failure_mode_effect \
    -te_name CMOS_28nm \
    -analysis_type permanent+transient \
    -parent_fmeda Monitoring_FMEDA \
    -failure_mode_effect_name Signal_Drift_Effect \
    -sreq_name {SREQ_MON_005 SREQ_MON_012} \
    -fr_value 6.15 \
    -user_defined_arguments {}
```

5.6.16 define_metric_iso26262

Table 31—define_metric_iso26262 command

Purpose	Return a metric value for a metric type defined in ISO 26262
Syntax	**define_metric_iso26262** **-metric_name** <**spfm** | **lfm** | **pmhf** > **-dc_type** <**estimated** | **measured** > **-fmeda_type** <**assumption-based** | **calculation-based** > **-scope** <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** > **-te_name** *te_name* **-analysis_type** <**permanent** | **transient** | **permanent+transient** > **-parent_fmeda** *parent_fmeda* [**-fmeda_name** *fmeda_name*] **-element_name** *element_name* **-failure_mode_name** *failure_mode_name* **-failure_mode_effect_name** *failure_mode_effect_name* **-parent_element** *parent_element* [**-sreq_name** *sreq_name_list*] **-metric_value** *float* [*0.0*, *n*] [**-user_defined_arguments** *list_of_tuples*]

Arguments	**-metric_name** <**spfm** | **lfm** | **pmhf** >	The metric type calculated according to ISO 26262
	-dc_type <**estimated** | **measured** >	The diagnostic coverage type used for reporting the estimated or measured metric
	-fmeda_type <**assumption-based** | **calculation-based** >	The type of FMEDA used to calculate the size of elements and failure modes
	-scope <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** >	The scope for which the metric is calculated
	-te_name *te_name*	The technology element for which the metric is calculated
	-analysis_type <**permanent** | **transient** | **permanent+transient** >	The analysis type for which the metric is calculated
	-parent_fmeda *parent_fmeda*	The FMEDA within which the metric is calculated; identifies the active FMEDA context for the operation
	-fmeda_name *fmeda_name*	When **-scope** is **fmeda** and the *FMEDA* is hierarchical, the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the metric is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*
	-element_name *element_name*	The name of the element in the FS hierarchy. *This argument is required only if -scope is element
	-failure_mode_name *failure_mode_name*	The name of the failure mode. *This argument is required only if -scope is failure_mode
	-failure_mode_effect_name *failure_mode_effect_name*	The name of the failure mode effect. *This argument is required only if -scope is failure_mode_effect
	-parent_element *parent_element*	The parent element in the FS hierarchy
	-sreq_name *sreq_name_list*	The safety requirement(s) for which the metric is calculated
	-metric_value *float* [*0.0*, *n*]	The value of the metric
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return the computed metric value as a float if successful, or raise a TCL_ERROR if not.	

define_metric_iso26262 command enables calculation and reporting of ISO 26262 metrics at various levels of the *FS hierarchy*, supporting both estimated and measured diagnostic coverage, and both assumption-based and calculation-based FMEDA methodologies.

define_metric_iso26262 command returns a metric value for a specified metric type as defined in ISO 26262.

- **-metric_name** — specifies the metric type to be calculated according to ISO 26262. Supported values include spfm (Single Point Fault Metric), lfm (Latent Fault Metric), and pmhf (Probabilistic Metric for random Hardware Failures).
- **-dc_type** — specifies whether the diagnostic coverage used for the metric calculation is estimated or measured.
- **-te_name** — specifies the *technology element* for which the metric is calculated.

- **-fmeda_type** — specifies the FMEDA methodology used to determine the size of *elements* and *failure modes*. Supported values are assumption-based and calculation-based. For assumption-based, user-assigned values take precedence; for calculation-based, design mapping-based values take precedence.
- **-analysis_type** — specifies the analysis type for which the metric is calculated. Supported values are permanent, transient, and permanent+transient, which sums over the metrics for both permanent and transient faults.
- **-parent_fmeda** — specifies the *FMEDA* within which the metric is calculated, identifying the active *FMEDA* context for the operation.
- **-fmeda_name** — when **-scope** is `fmeda` and the *FMEDA* is hierarchical, specifies the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the metric is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*.
- **-scope** — specifies the scope for which the metric is calculated. Supported values are fmeda, element, failure_mode, and failure_mode_effect.
- **-element_name** — specifies the name of the *element* in the *FS hierarchy*. This field is mandatory if **-scope** is element.
- **-failure_mode_name** — specifies the name of the *failure mode*. This field is mandatory if **-scope** is failure_mode.
- **-failure_mode_effect_name** — specifies the name of the *failure mode effect*. This field is mandatory if **-scope** is failure_mode_effect.
- **-parent_element** — specifies the identifier of the parent *element* to which the scope (*element*/*failure mode*) is connected in the *FS hierarchy*, enabling unique identification of the scope (*element*/*failure mode*). This field is mandatory if **-scope** is element or failure_mode.
- **-sreq_name** — specifies the list of *safety requirements* for which the metric is calculated. If not specified, the *failure mode* to *safety requirement* assignment is not considered in the metric calculation.
- **-metric_value** — specifies the value of the metric [unit: as defined by ISO 26262].

Rules

The following rules apply:

Table 32—Rules for ‘define_metric_iso26262’ command

Rule ID	Rule
Rule_Metric_ISO_1	When **-dc_type** is set to estimated, ISO 26262 metrics are calculated based on diagnostic coverage defined by one of the following: - **-dc** specified in **create_sm** (isolated DC) - **-dc_*_estimated** assigned by **assign_sm_fm** (assigned estimated DC) - **-dc_*_expert** specified in **create_fm** (expert DC) The following priority rules apply: - When both isolated DC and assigned estimated DC exist for the same object, assigned estimated DC shall take precedence over isolated DC. - When expert DC exists for the same object, expert DC shall take precedence over assigned estimated DC or isolated DC.NOTE: Aggregation rule "Rule_FMEDA_5" applies.

Rule_Metric_ISO_2	When **-dc_type** is set to measured, ISO 26262 metrics are calculated based on diagnostic coverage defined by one of the following: - **-dc_*_measured** assigned by **assign_sm_fm** (assigned measured DC) - **-dc_*_measured** specified in **create_fm** (aggregated over the *safety mechanisms* associated with the *failure mode*)When **-dc_measured** is not available, the command shall fall back to either isolated DC, assigned estimated DC, or expert DC, as defined in Rule_Metric_ISO_1. NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_Metric_ISO_3	When both user-defined area estimations (FM_size_*_assumption_based) and design mapping exist for the same *failure mode*, the following shall apply for ISO 26262 metrics calculation: - If **-fmeda_type** is set to assumption-based: FM_size_*_assumption_based shall take precedence over FM_size_*_calculation_based. - If **-fmeda_type** is set to calculation-based: FM_size_*_calculation_based shall take precedence over FM_size_*_assumption_based.When only one of the two sources is available, the available value shall apply.
Rule_Metric_ISO_4	When ISO 26262 metrics are reported with **-sreq_name** specified, the following shall apply: A *failure mode* shall be considered safety relevant if it is classified as safety relevant for at least one of the specified *safety requirements*.
Rule_Metric_ISO_5	When ISO 26262 metrics are reported without specifying **-sreq_name**, the following shall apply: Metrics shall be reported independently of *safety requirements*.

How-to Examples

Overview

The define_metric_iso26262 command calculates and returns ISO 26262 safety metrics at various levels of the functional safety hierarchy. It supports SPFM (Single Point Fault Metric), LFM (Latent Fault Metric), and PMHF (Probabilistic Metric for Hardware Failure) calculations with both estimated and measured diagnostic coverage for automotive semiconductor applications.

Example 1: FMEDA-Level SPFM Calculation for Automotive Gateway SoC

This example calculates the Single Point Fault Metric at the FMEDA level for an automotive gateway SoC using estimated diagnostic coverage and assumption-based FMEDA methodology to verify ASIL-D compliance for vehicle networking applications.

```
define_metric_iso26262 \
    -metric_name spfm \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_16nm \
    -analysis_type permanent \
    -parent_fmeda Gateway_FMEDA \
    -metric_value 0.97 \
    -user_defined_arguments {}
```

Example 2: Element-Level LFM Calculation for ADAS Processing IP

This example calculates the Latent Fault Metric for an ADAS image processing IP element within an automotive vision SoC using measured diagnostic coverage from hardware fault injection and calculation-based FMEDA for ASIL-B image recognition systems.

```
define_metric_iso26262 \
    -metric_name lfm \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope element \
    -te_name CMOS_7nm \
    -analysis_type permanent \
    -parent_fmeda ADAS_Vision_FMEDA \
    -element_name Image_Processor \
    -parent_element ADAS_Vision_SoC \
    -metric_value 0.92 \
    -user_defined_arguments {}
```

Example 3: FMEDA-Level PMHF Calculation for Powertrain Control SoC

This example calculates the Probabilistic Metric for Hardware Failure combining both permanent and transient faults at the FMEDA level for an engine control SoC using measured diagnostic coverage to verify ASIL-D compliance for powertrain safety functions.

```
define_metric_iso26262 \
    -metric_name pmhf \
    -dc_type measured \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_28nm \
    -analysis_type permanent+transient \
    -parent_fmeda Engine_Control_FMEDA \
    -metric_value 8.5e-9 \
    -user_defined_arguments {}
```

Example 4: Failure Mode Level SPFM for Battery Management IP

This example calculates SPFM at the failure mode level for a battery management IP within an electric vehicle power management SoC, analyzing charge controller failure modes for ASIL-C battery safety monitoring systems.

```
define_metric_iso26262 \
    -metric_name spfm \
    -dc_type estimated \
    -fmeda_type calculation-based \
    -scope failure_mode \
    -te_name CMOS_40nm \
    -analysis_type permanent \
    -parent_fmeda Battery_Mgmt_FMEDA \
    -element_name Charge_Controller \
    -failure_mode_name Overcharge_Detection_Fault \
    -parent_element Battery_Mgmt_SoC \
    -metric_value 0.95 \
    -user_defined_arguments {}
```

Example 5: Granular Failure Mode Effect LFM for Radar Sensor IP

This example demonstrates failure mode effect level LFM calculation for a radar sensor IP within an

autonomous driving SoC, analyzing range measurement degradation effects for ASIL-D autonomous emergency braking with comprehensive diagnostic coverage and safety requirement filtering.

```
define_metric_iso26262 \
    -metric_name lfm \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope failure_mode_effect \
    -te_name CMOS_16nm \
    -analysis_type permanent+transient \
    -parent_fmeda Radar_FMEDA \
    -failure_mode_effect_name Range_Degradation_Effect \
    -sreq_name {SREQ_RADAR_008 SREQ_RADAR_015 SREQ_AEB_003} \
    -metric_value 0.88 \
    -user_defined_arguments {}
```

5.6.17 define_metric_iec61508

Table 33—define_metric_iec61508 command

Purpose	Return a metric value for a metric type defined in IEC 61508	
Syntax	**define_metric_iec61508** **-metric_name** <**sff** | **probability_dangerous_failure_low_demand** | **probability_dangerous_failure_high_demand** > **-dc_type** <**estimated** | **measured** > **-fmeda_type** <**assumption-based** | **calculation-based** > **-scope** <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** > **-te_name** *te_name* **-analysis_type** <**permanent** | **transient** | **permanent+transient** > **-parent_fmeda** *parent_fmeda* [**-fmeda_name** *fmeda_name*] **-element_name** *element_name* **-failure_mode_name** *failure_mode_name* **-failure_mode_effect_name** *failure_mode_effect_name* **-parent_element** *parent_element* [**-sreq_name** *sreq_name_list*] **-metric_value** *float* [*0.0*, *n*] [**-user_defined_arguments** *list_of_tuples*]	
Arguments	**-metric_name** <**sff** | **probability_dangerous_failure_low_demand** | **probability_dangerous_failure_high_demand** >	The metric type calculated according to IEC 61508
	-dc_type <**estimated** | **measured** >	The diagnostic coverage type used for reporting the estimated or measured metric
	-fmeda_type <**assumption-based** | **calculation-based** >	The type of FMEDA used to calculate the size of elements and failure modes
	-scope <**fmeda** | **element** | **failure_mode** | **failure_mode_effect** >	The scope for which the metric is calculated
	-te_name *te_name*	The technology element for which the metric is calculated
	-analysis_type <**permanent** | **transient** | **permanent+transient** >	The analysis type for which the metric is calculated

	-parent_fmeda *parent_fmeda*	The FMEDA within which the metric is calculated; identifies the active FMEDA context for the operation
	-fmeda_name *fmeda_name*	When **-scope** is **fmeda** and the *FMEDA* is hierarchical, the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the metric is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*
	-element_name *element_name*	The name of the element in the FS hierarchy. *This argument is required only if -scope is element
	-failure_mode_name *failure_mode_name*	The name of the failure mode. *This argument is required only if -scope is failure_mode
	-failure_mode_effect_name *failure_mode_effect_name*	The name of the failure mode effect. *This argument is required only if -scope is failure_mode_effect
	-parent_element *parent_element*	The parent element in the FS hierarchy
	-sreq_name *sreq_name_list*	The safety requirement(s) for which the metric is calculated
	-metric_value *float* [*0.0*, *n*]	The value of the metric
	-user_defined_arguments *list_of_tuples*	A list of user-defined arguments and their values
Return value	Return the computed metric value as a float if successful, or raise a TCL_ERROR if not.	

define_metric_iec61508 command enables calculation and reporting of IEC 61508 metrics at various levels of the *FS hierarchy*, supporting both estimated and measured diagnostic coverage, and both assumption-based and calculation-based FMEDA methodologies.

define_metric_iec61508 command returns a metric value for a specified metric type as defined in IEC 61508.

- **-metric_name** — specifies the metric type to be calculated according to IEC 61508. Supported values include sff (Safe Failure Fraction), probability_dangerous_failure_low_demand, and probability_dangerous_failure_high_demand.
- **-dc_type** — specifies whether the diagnostic coverage used for the metric calculation is estimated or measured.
- **-te_name** — specifies the *technology element* for which the metric is calculated.
- **-fmeda_type** — specifies the FMEDA methodology used to determine the size of *elements* and *failure modes*. Supported values are assumption-based and calculation-based. For assumption-based, user-assigned values take precedence; for calculation-based, design mapping-based values take precedence.
- **-analysis_type** — specifies the analysis type for which the metric is calculated. Supported values are permanent, transient, and permanent+transient, which sums over the metrics for both permanent and transient faults.
- **-parent_fmeda** — specifies the *FMEDA* within which the metric is calculated, identifying the active *FMEDA* context for the operation.
- **-fmeda_name** — when **-scope** is `fmeda` and the *FMEDA* is hierarchical, specifies the name of a specific child *FMEDA* instance within **-parent_fmeda** for which the metric is calculated; if omitted, the result spans all child *FMEDAs* within the *parent_fmeda*.
- **-scope** — specifies the scope for which the metric is calculated. Supported values are fmeda, element, failure_mode, and failure_mode_effect.

- **-element_name** — specifies the name of the *element* in the *FS hierarchy*. This field is mandatory if **-scope** is element.
- **-failure_mode_name** — specifies the name of the *failure mode*. This field is mandatory if **-scope** is failure_mode.
- **-failure_mode_effect_name** — specifies the name of the *failure mode effect*. This field is mandatory if **-scope** is failure_mode_effect.
- **-parent_element** — specifies the identifier of the parent *element* to which the scope (*element*/*failure mode*) is connected in the *FS hierarchy*, enabling unique identification of the scope (*element*/*failure mode*). This field is mandatory if **-scope** is element or failure_mode.
- **-sreq_name** — specifies the list of *safety requirements* for which the metric is calculated. If not specified, the *failure mode* to *safety requirement* assignment is not considered in the metric calculation.
- **-metric_value** — specifies the value of the metric [unit: as defined by IEC 61508].

Rules

The following rules apply:

Table 34—Rules for 'define_metric_iec61508' command

Rule ID	Rule
Rule_Metric_IEC_1	When **-dc_type** is set to estimated, IEC 61508 metrics are calculated based on diagnostic coverage defined by one of the following: - **-dc** specified in **create_sm** (isolated DC) - **-dc_*_estimated** assigned by **assign_sm_fm** (assigned estimated DC) - **-dc_*_expert** specified in **create_fm** (expert DC) The following priority rules apply: - When both isolated DC and assigned estimated DC exist for the same object, assigned estimated DC shall take precedence over isolated DC. - When expert DC exists for the same object, expert DC shall take precedence over assigned estimated DC or isolated DC.NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_Metric_IEC_2	When **-dc_type** is set to measured, IEC 61508 metrics are calculated based on diagnostic coverage defined by one of the following: - **-dc_*_measured** assigned by **assign_sm_fm** (assigned measured DC) - **-dc_*_measured** specified in **create_fm** (aggregated over the *safety mechanisms* associated with the *failure mode*)When **-dc_measured** is not available, the command shall fall back to either isolated DC, assigned estimated DC, or expert DC, as defined in Rule_Metric_IEC_1. NOTE: Aggregation rule "Rule_FMEDA_5" applies.
Rule_Metric_IEC_3	When both user-defined area estimations (FM_size_*_assumption_based) and design mapping exist for the same *failure mode*, the following shall apply for IEC 61508 metrics calculation: - If **-fmeda_type** is set to assumption-based: FM_size_*_assumption_based shall take precedence over FM_size_*_calculation_based. - If **-fmeda_type** is set to calculation-based: FM_size_*_calculation_based shall take precedence over FM_size_*_assumption_based.When only one of the two sources is available, the available value shall apply.
Rule_Metric_IEC_4	When IEC 61508 metrics are reported with **-sreq_name** specified, the following shall apply: A *failure mode* shall be considered safety relevant if it is classified as safety relevant for at least one of the specified *safety requirements*.
Rule_Metric_IEC_5	When IEC 61508 metrics are reported without specifying **-sreq_name**, the following shall apply: Metrics shall be reported independently of *safety requirements*.

How-to Examples

Overview

The define_metric_iec61508 command calculates and returns IEC 61508 safety metrics at various levels of the functional safety hierarchy. It supports SFF (Safe Failure Fraction), PFD (Probability of Dangerous Failure) for low demand and high demand modes with both estimated and measured diagnostic coverage for industrial safety-critical semiconductor systems.

Example 1: FMEDA-Level SFF Calculation for Industrial PLC SoC

This example calculates the Safe Failure Fraction at the FMEDA level for an industrial programmable logic controller SoC using estimated diagnostic coverage and assumption-based FMEDA methodology to verify SIL-3 compliance for process control automation.

```
define_metric_iec61508 \
    -metric_name sff \
    -dc_type estimated \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_28nm \
    -analysis_type permanent \
    -parent_fmeda PLC_Control_FMEDA \
    -metric_value 0.94 \
    -user_defined_arguments {}
```

Example 2: Element-Level PFD Low Demand for Safety Interlock IP

This example calculates the Probability of Dangerous Failure in low demand mode for a safety interlock IP element within an emergency shutdown system SoC using measured diagnostic coverage from FPGA emulation and calculation-based FMEDA for SIL-2 interlock functions.

```
define_metric_iec61508 \
    -metric_name probability_dangerous_failure_low_demand \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope element \
    -te_name CMOS_40nm \
    -analysis_type permanent \
    -parent_fmeda ESD_System_FMEDA \
    -element_name Safety_Interlock \
    -parent_element ESD_System_SoC \
    -metric_value 2.5e-4 \
    -user_defined_arguments {}
```

Example 3: FMEDA-Level PFD High Demand for Burner Management SoC

This example calculates the Probability of Dangerous Failure in high demand mode combining both permanent and transient faults at the FMEDA level for a burner management system SoC using measured diagnostic coverage to verify SIL-3 compliance for continuous combustion control.

```
define_metric_iec61508 \
    -metric_name probability_dangerous_failure_high_demand \
```

```
    -dc_type measured \
    -fmeda_type assumption-based \
    -scope fmeda \
    -te_name CMOS_65nm \
    -analysis_type permanent+transient \
    -parent_fmeda Burner_Mgmt_FMEDA \
    -metric_value 1.8e-7 \
    -user_defined_arguments {}
```

Example 4: Failure Mode Level SFF for Pressure Sensor Interface IP

This example calculates SFF at the failure mode level for a pressure sensor interface IP within an industrial monitoring SoC, analyzing sensor signal conditioning failure modes for SIL-2 pressure monitoring in chemical process systems.

```
define_metric_iec61508 \
    -metric_name sff \
    -dc_type estimated \
    -fmeda_type calculation-based \
    -scope failure_mode \
    -te_name CMOS_28nm \
    -analysis_type permanent \
    -parent_fmeda Pressure_Monitor_FMEDA \
    -element_name Sensor_Interface \
    -failure_mode_name Signal_Conditioning_Fault \
    -parent_element Pressure_Monitor_SoC \
    -metric_value 0.91 \
    -user_defined_arguments {}
```

Example 5: Granular Failure Mode Effect PFD for Valve Actuator Controller IP

This example demonstrates failure mode effect level PFD low demand calculation for a valve actuator controller IP within an industrial safety shutdown SoC, analyzing valve position feedback loss effects for SIL-3 emergency isolation valves with comprehensive diagnostic coverage and safety requirement filtering.

```
define_metric_iec61508 \
    -metric_name probability_dangerous_failure_low_demand \
    -dc_type measured \
    -fmeda_type calculation-based \
    -scope failure_mode_effect \
    -te_name CMOS_40nm \
    -analysis_type permanent+transient \
    -parent_fmeda Valve_Control_FMEDA \
    -failure_mode_effect_name Feedback_Loss_Effect \
    -sreq_name {SREQ_VALVE_003 SREQ_VALVE_009 SREQ_ESD_012} \
    -metric_value 1.2e-4 \
    -user_defined_arguments {}
```

Annex A

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] Accellera Functional Safety White Paper (May 2021). [7]

[B2] Accellera Functional Safety White Paper (December 2023). [8]

[B3] IEEE Standards Dictionary Online.[9]

[B4] ISO/IEC 8859-1, Information technology—8-bit single-byte coded graphic character sets—Part 1: Latin Alphabet No. 1.[10]

[B5] Tcl language syntax summary.[11]

[B6] Tcl language usage.[12]

[B7] OMG SysML v1.7, Systems Modeling Language Specification, Object Management Group, 2024.[13]

[B8] OMG SysML v2.0, Systems Modeling Language (SysML) Version 2.0 Specification, Object Management Group, 2025.[14]

[B9] IEEE 1666-2023, IEEE Standard SystemC Language Reference Manual, IEEE Standards Association, 2023.[15]

[B10] IEEE 1364-2005, IEEE Standard for Verilog Hardware Description Language, IEEE Standards Association, 2005. [16]

[B11] IEEE 1076-2019, IEEE Standard VHDL Language Reference Manual, IEEE Standards Association, 2019.[17]

[B12] Berkeley SPICE, SPICE2: A Computer Program to Simulate Semiconductor Circuits, University of California, Berkeley, 1975 (and subsequent SPICE3 releases).

[B13] IEEE 1685-2022, IP-XACT Standard Structure for Packaging, Integrating, and Reusing IP within tool flows.[18]

---

[7] Available at https://www.accellera.org/images/downloads/standards/functional-safety/Functional_Safety_White_Paper_051020.pdf
[8] Available at https://www.accellera.org/images/downloads/standards/functional-safety/Functional_Safety_White_Paper_20231213.pdf
[9] Available at http://www.ieee.org/publications_standards/publications/subscriptions/prod/standards_dictionary.html
[10] ISO/IEC publications are available from the ISO Central Secretariat (http://www.iso.org/). ISO publications are also available in the United States from the American National Standards Institute (http://www.ansi.org/).
[11] Available at http://www.tcl.tk/man/tcl8.4/TclCmd
[12] Available at http://sourceforge.net/projects/tcl/
[13] OMG publications are available from the Object Management Group® Standards Development Organization (https://www.omg.org/technology/documents/)
[14] See Footnote 12.
[15] IEEE publications are available from The Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).
[16] See Footnote 13.
[17] See Footnote 13.
[18] See Footnote 13.