

Carbon-ML – The Language of Carbon – White Paper DRAFT

Version: 2.2 updated July 28, 2022

Author(s): Lynn Connolly, lynn.connolly@carbon-ml.org
Rene Monroy, rene.monroy@carbon-ml.org
Nick Gogerty, ng@carbonfinancelab.com
Jonathan Hollander, jh@carbonfinancelab.com

Goal of document: An overview of Carbon-ML ecosystem, < Carml> language features and functions use cases / benefits and governance suggestions.

Using this document: This document has high level examples and business cases and is not intended to be overly proscriptive or exhaustive of implementations or states, tools, actors or deployments.

Fully enumerated <CaRML> examples and detailed reference implementation are kept in an online [Github Repository](#).

Table of Contents

About Us	5
The Vision	6
Operationalizing a Vision	7
The Motivation is Faster High Quality Data Flow	7
Adaptation for Embodied Carbon	8
Our Goals and Outcomes	9
What is the Carbon-ML Ecosystem	10
The Carbon-ML Ecosystem:	10
Proposed Carbon-ML Ecosystem Consortia structure	11
Carbon-ML Ecosystem Consortium Members (participants)	11
Consortium Summary Goals:	11
Carbon-ML Ecosystem Consortium Members and Carbon-ML.org Commitment:	11
Carbon-ML Ecosystem Consortium Members Summary of Provided Expertise:	13
Global Organizations Providing Expertise and Standards for Multiple Industries	13
Industry Companies and Associations	13
Industry Oversight Organizations	13
Existing Embodied Carbon Working Groups, Academics and Others	13

Regulatory and Governance Oversight	13
Technology Experts/Organizations	13
Technical Design Architects & Ecosystem Thinkers	14
Carbon-ML Ecosystem Consortia Governance Framework	15
Carbon-ML Ecosystem Consortia Schema and Taxonomies	16
Carbon-ML Ecosystem Schema (Branch) and Taxonomy (Node) Principles:	17
Extensibility and Reference Schema principles:	17
Schema related Taxonomy = Intelligent Context:	17
Customizable for Regional and Local Variants:	17
Carbon-ML Ecosystem Standard CO2e Declaration Language <CarML>	18
What is <CarML>	18
Carbon-ML Ecosystem CO2e <CarML> Message Type	20
What is Meant by a <CarML> Message Type?	20
<CarML> Message Type Root Framework	21
<CarML> PoC Message Type General	22
<CarML> Message Type Declaration Input Descriptions	22
General (Name)	23
Why (To Be Added) message intent	23
What (Unique ID)	23
Amount (Amount)	24
Who (Declared Trait Data Source)	24
How (Declared Trait Measurement Methodology)	24
When (Date and Time)	25
Where (Location)	25
Verifier (Verifying Entity)	25
Environmental Mitigation Instruments (Carbon Credit)	25
<CarML> Message Types Use Case Examples	26
Summary Use cases for users/consumers of <CaRML> messages	26
Producers/publishers of <CaRML> message types can include:	26
<CarML> Message Type Category Examples in Support of Use Cases	27
Consumer Goods (example)	27
Manufacturing (example)	29
Regulatory Requirements (example)	30
Environmental Mitigation Instruments RFQ (such as Carbon Credit Request for Quote) (Example)	31

Financial Markets Environmental Risk (example)	32
Transportation Services (example)	33
<CarML> Message Types Formal vs. Informal	35
APPENDIX A: <CarML> Message Type Technical Design	36
<CarML> Message Type Technical Design Summary	37
<CarML> Technical Decisions Made and Summary of Thought Process	39
Role and Responsibility of Carbon-ML.org in the Technical Development of <CarML>	39
Clean Architecture	39
Domain Driven Design	39
JSON vs. XML	39
A new markup/tag structure	39
HTML and Desktop Apps	39
<CarML> Message Type Generator Engineering Architecture	40
<CarML> Message Type Generator Interface Backend Database Architecture	40
Very Basic Database Architecture Design example	40
<CarML>'s Added Complexity to Database Architecture Design	43
<CarML> Message Type Backend Architecture Design Solution	44
Development and design of a <CarML> Unique ID Summary	45
Step One: Development of the Product and/or Service Unique <CarML> ID Mapping Table	45
<CarML> Unique ID Generator Toolkit Summary	45
Company Profile Flow	48
Step 2: Mapping of product and/or service CO2e declaration to the corresponding product and/or service unique <CarML> ID	51
Step 3: <CarML> frontend fetching, using pre-defined <CarML> "tags", the required data for the <CarML> Message Type from the backend databases by going through the <CarML> Domain/Translation Layer which defines business objects/data entities/use cases and essentially "mapping" of the data.	52
<CarML> Message Type Generator Interface data fields technical flow summary	52
<CarML> Message Type Generator – The Frontend User Interface Engineering Architecture	53
Views	54
Components	54
Store	54
Services	55
Summarized Workflow Example	55
<CarML> PoC Message Type General Form	56

External Libraries	57
Form Lifecycle	57
Workflow Example	57
Overall <CarML> Message Type Generator PoC Frontend Architecture	58
Web Application	58
Repository	59
CI/CD	59
RESTful API	60
Clean Architecture Principles for <CarML> Development Recommendation	61
The <CarML> Clean Architecture Domain Layer	63
<CarML> Message Type Compliance Check Process	65
<CarML> Message Type FAQs	66
Appendix B: Carbon-ML Ecosystem Initial Schemas and Taxonomies Categories and Industries	67
Appendix C: Carbon-ML Ecosystem Definition of Terms (technical and business terms)	68
Appendix D: Carbon-ML.org Guiding Principles	70

About Us

The Carbon-ML project is developing an open-source ecosystem and language to support ongoing declarations of embodied carbon measurements in any product or service using global and local contexts.

Carbon-ML is incubated by [Carbon Finance Labs](#) in partnership with [Oxy Low Carbon Ventures](#). The goal is for Carbon-ML to evolve into an independently governed project with multiple private, public sectors actors spanning industries, technology providers, data consumers and geographies.



Carbon Finance Labs

A finance and technology incubator creating climate change solutions. Our impact comes from a global network of resources and knowledge built over decades spent in the carbon, finance and technology sectors.



L&W CARBON
VENTURES

Oxy Low Carbon Ventures, LLC (OLCV), a subsidiary of Occidental, Petroleum

The Vision

As the world becomes increasingly concerned with carbon emissions accelerating environmental and climate impacts, there is an increasing emphasis on understanding the amount of carbon equivalent¹ emissions produced during the entire lifecycle of a product or service.

Therein lies a problem. Data about the embodied carbon in the buying, producing or selling products or services is hidden and stuck across supply chains. Currently, the measuring, reporting, tracking and declaration of embodied carbon within a product or service is a mess and doesn't flow. There are numerous measurements, policies, mandates, systems, etc. designed to measure carbon emissions, but lack of consistency between them hinders carbon data flow. These systems and metrics are designed to report carbon at the company level, not at the product or service level.

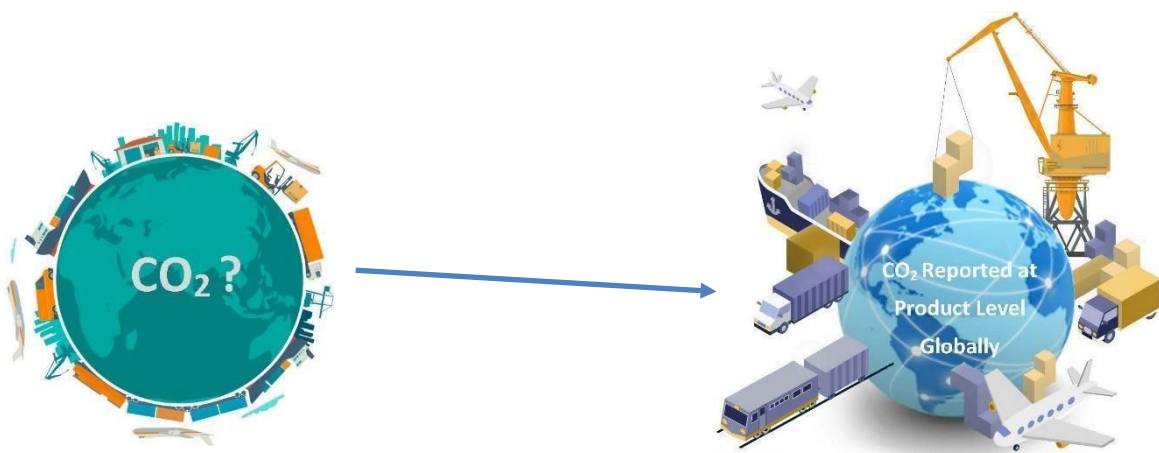
Our vision is for embodied carbon data to freely flow between all product or service related system interactions across supply chains, while maintaining useful contexts. Everywhere there is a price tag, there should be a digital embodied carbon (CO₂e²) tag equivalent.

This can be created by:

- developing an open-source global ecosystem of actors:
- an extensible declarative schema using existing product and service taxonomies,
- enabling trusted and visible declarations of embodied carbon
- across all products or services at all points and actors across supply chains.

The target industries, products, services, and users include everyone – all industries, products and services; and related inventors, designers, developers, manufacturers, wholesale purchasers, sellers, buyers, consumers, monitors, regulators, transporters, researchers, etc. The world.

Our goal is ever more useful standardized contexts for embodied carbon where the embodied product carbon is declared with standardized context understandable at any point by any person.



¹ Many gases contribute to green house gas effects. Effects are normalized to CO₂e (carbon dioxide equivalents)

² Generally speaking in this document when “carbon is mentioned” it should be considered to be CO₂e, the GWP of related gases all normalized to CO₂e using the appropriate GHG GWP convention.

Operationalizing a Vision

Tackling the world's products and services, and every person interacting with a product or service at every point along the supply chain is an enormous undertaking. A grandiose vision that requires a scaled, thoughtful, and phased approach.

Making this a reality means incorporating the work of others and building an open-source evolving solution. We work with others to create consortia of like-minded actors each with roles, use cases and responsibilities to provide input on development, usage and acceptance for fit for purpose solutions.

The first issue/open question is: 'Is there a systems language to provide context for something that can be easily adopted rapidly and globally.'

We found our answer by looking at the world of accounting and business reporting for inspiration.

The Motivation is Faster High Quality Data Flow

The world of accounting and business reporting of economic flows has [XBRL](#) and [iXBRL](#), an open international standard for digital business reporting, managed by a global not for profit consortium, XBRL International.

XBRL is an open-source, freely available global framework of accounting standards used for the exchange of business information. XBRL was developed in 1998 with the latest version introduced in 2003. It continues to evolve as additional XBRL modules, tools and language extensions are developed delivering new functionality and use cases. XBRL is open-source, scalable, extensible, adaptable for countries and/or regions, continues to evolve and has widespread global adoption in private and public sectors.

Part of its functionality is in enabling the flow of automated reporting, data sharing, regulatory filings, etc. and reduction of bad data, and radically shrinking the need for conversions and integrations etc. In short, XBRL accelerates economic activity and data flows by structuring accounting data and contexts.

As an example, accounting concepts such as depreciation, goodwill and intangible assets among others are interpreted or presented in a standardized context relative to accounting frameworks at the declared presentation layer. The core data values are clearly declared and transparent, but the resultant displayed interpretation may vary based on reporting or other requirements such as GAAP v. IFRS accounting standards for example.

Adaptation for Embodied Carbon

Embodied Carbon (in this document the phrase ‘embodied carbon’ represents the sum total CO₂e of any related GHG gasses which were added during economic value-added processes) currently has no formal language of **accounted** or declared expression for items / tags that would provide shared context such as:

- Why the embodied carbon declaration was made (intended uses),
- Who (a legal entity such as a corporation often times referred to as a reference entity) made the embodied carbon declaration and who verified it,
- The quality of measurement (uncertainty) associated with data and measurement,
- What is being declared (product or service),
- How much embodied carbon is being declared,
- When was the embodied carbon declaration made,
- Where was the embodied carbon declaration made,
- Are there other embodied carbon factors that should be contextualized.

In addition, current fuzzy marketing concepts such as “green,” “carbon neutral,” etc. without being formally defined are used to denote multiple activities. This causes issues in determining the quantitative benefits or impacts when assessing purchasing decisions for industrial or individual consumption or compliance across a supply chain that more formalized attestations or declarations could provide.

Compounding this are attempts to establish carbon offset programs related to the embodied carbon of a product or service, where today services such as bundling credits/offsets or other market-based tools are only informally referred to as “tonnes” of X.

A formal declarative language and context of embodied carbon processes allows formal attestation to be made and managed in structured but open-source schema, similar to how XBRL evolved. It is hoped that all parties across supply chains can declare in a standardized context the carbon associated with their purchasing, value add and selling activities.

We believe that transparent CO₂e signaling which is standardized, machine readable, and extensible can accelerate coordinated choices, actions, and consensus on normative activities such as reporting across supply chains. This data can initially create awareness of embodied carbon and then millions or billions of deeper actions and choices to mitigate and reduce CO₂e as efforts to create new higher valued products or services become more certain financially, reputationally, and from regulatory reporting perspectives.

Our Goals and Outcomes

Our goal is to make embodied carbon data flows related to declaring & sharing easy.

This likely requires:

1. An evolving ecosystem of actors.
2. Systems comprised of extensible schema referencing existing and evolving product or service taxonomies.
3. An ecosystem that is trusted and visible, open-source, adaptable for easy implementation, globally adoptable, and technology agnostic.
4. Creating structured data declarations of measured embodied carbon (CO₂e).

This enables the creation of:

1. Structured data declarations of measured CO₂e.
2. Supporting every product or service at any point along the supply chain – upstream and downstream.
3. Information about the why, who, what, how, when, and where of the CO₂e declaration, among other data reference points.

Empowering:

Private and public sector actors that produce, consume, govern, and track declarations of CO₂e with maintainable, reasonable and standardized context for any product or service across supply chains.

Ecosystem partners to develop carbon related message types and related solutions such as displaying a product's or service's CO₂e on the product's or service's label/description so that companies, wholesalers, consumers, suppliers, governments, etc. can make informed choices.

Possible outcomes from structured data declarations of CO₂e in products and services may include:

- **Influencing major actor purchasing changes** along the supply chains such as corporate purchasing, customer pressure, and government policy.
- **Encouraging new behaviors** through better understanding of CO₂e, conversations and trust, and resulting in millions of new choices being made.
- **Having economic impacts** with new goods & services to drive growth in the voluntary carbon market.
- **Empowering Buy Clean initiatives** beyond those drivers of Buy Clean initiators.

What is the Carbon-ML Ecosystem

Definition of ecosystem from Merriam-Webster.com/dictionary/ecosystem, May 2022:

- 1) The complex of a community of organisms and its environment functioning as an ecological unit in nature.*
- 2) Something (such as a network of businesses) considered to resemble an ecological ecosystem especially because of its complex interdependent parts.*

The Carbon-ML Ecosystem:

The Carbon-ML Ecosystem is a purposeful business arrangement between a collection of industry, services, and regulatory consortia to develop a standardized language and message type formats for accelerating the declaration of CO₂e for any product and/or service at any point along a supply chain.

Carbon-ML is envisioned as an evolving ecosystem of:

- **Global industry and services participants** along with regulatory consortia, (private companies, govt, regulators, NGO etc.)
- **Extensible schemas** of taxonomies maintained & versioned in an open but rigorous way
- **Taxonomies of data** & contexts, from existing sectors & verticals
- **Open mix of technology, languages, tools and platforms** for expression, usage and management
- **Governance principles** for organization & growth to maximize data flow & message type usage

A [GitHub site for Carbon-ML](#) is used for collaborative development of the Carbon-ML ecosystem, building consensus with existing schemas and taxonomies, and eventually realizing a globally adoptable standard declarative language for CO₂e measurement declarations that is technology agnostic.

The ecosystem consortia each form consensus-based decisions around the use of extensible schemas supported by existing product or service taxonomies, or extended/developed from scratch (only when needed). These decisions inform outcomes enabling standardized, trusted and visible declarations of CO₂e in every product or service where standardized CO₂e declaration context is maintained across the supply chain.

The primary open-source technical markup language to be used within the Carbon-ML ecosystem is <CarML>. However, the Carbon-ML ecosystem consortia can each accept multiple technical languages as long as the resultant standardized CO₂e declaration front-end display/label and back-end taxonomy (context) reference is maintained.

Proposed Carbon-ML Ecosystem Consortia structure

*Definition of consortium from Merriam-Webster.com/dictionary/consortium, May 2022:
An agreement, combination, or group (as of companies) formed to undertake an enterprise beyond the resources of any one member.*

The proposed Carbon-ML Ecosystem includes consortia from distinct industry verticals, and from service organizations that provide leading practices and standards across multiple industries and services such as standards and taxonomy providers, technology service providers, regulators, NGOs etc.

Each industry and/or sub-industry has its own Carbon-ML Ecosystem Consortium, as it is recognized that there are differences in the CO2e declaration reporting requirements by industry.

For organizations that provide leading practices and standards across multiple industries and services, they will be asked to be active participants in either their own service based consortia and/or multiple industry consortia to provide their leadership and expertise in these areas.

Carbon-ML Ecosystem Consortium Members (participants)

Consortium Summary Goals:

The goal of each Carbon-ML Ecosystem **Industry Consortium** is to form a consensus agreement as to the standard CO2e declaration message types to be used within their industry for the products or services at any point along the supply chain. The Carbon-ML Ecosystem **Service Organization Consortia** provides the Industry Consortia with the information/data factors they require to develop consistent, standardized CO2e declaration message types for use.

Existing schema and taxonomies will be used as well as the incorporation of regulatory requirements, etc. Developed standards and context definitions can be maintained at a centralized or decentralized entity as needed.

Governance structures and guiding principles are to be developed. The maintenance of such can be centralized or decentralized as needed.

All is to be open-source and transparent. Both formal and informal CO2e message types can be supported.

Carbon-ML Ecosystem Consortium Members and Carbon-ML.org Commitment:

Carbon-ML.org will:

- Collaborate on assembling industry consortia and determining roles and responsibilities
- Provide overall guidance and project management for the Carbon-ML project, including the development of CO2e declaration message types using <CarML>
- Ensure core principles and governance framework is followed
- Organize and facilitate consortia working sessions
- Ensure transparency and open-source policies are maintained

- Provide the necessary technical and managerial resources as needed to complete basic technical requirements and PoCs to demonstrate usability
- Maintain the overall Carbon-ML and <CarML> GitHub repository, technical architecture specs, required data dictionary and documentation, frontend to backend design and flows, among others
- Ensure that participant databases, message types, and other information remain private, when required

Carbon-ML Ecosystem Consortium Members will:

- Collaborate with Carbon-ML.org in determining the Carbon-ML ecosystem consortium they would like to participate in as a partner, lead advisor, contributor, commentor, observer, etc. and provide assistance with advising, as required, on the:
 - Development,
 - Promotion,
 - Additional participants, and
 - Determination of roles and responsibilities
- Provide expertise and work in shaping the resultant CO2e declaration Message Types
- Be an active in working sessions and CO2e declaration message type development using <CarML>,
- Provide guidance on governance, existing taxonomies and structures, and incorporation of such
- Assist in shaping terms, reference data to be sourced, definition of data fields, among others
- Lend expertise, provide use cases, and assist in the development of relevant CO2e declaration message types utilizing <CarML>
- Facilitate the use of Carbon-ML ecosystem consortium developed CO2e declaration message types using <CarML>
- May assist in marketing and promotion efforts, engage others

Carbon-ML Ecosystem Consortium Members Summary of Provided Expertise:

Global Organizations Providing Expertise and Standards for Multiple Industries

- Provide leadership and guiding principles
- Create use cases and best practices
- Contribute taxonomies, schema, datasets
- Design of CO2e declaration message types utilizing <CarML>

Industry Companies and Associations

- Provide leadership and guidance on schema, taxonomies, datasets
- Design and develop agreement on CO2e declaration message types utilizing <CarML> to share/consume
- Create use cases/best practice examples

Industry Oversight Organizations

- Contribute taxonomies, extension suggestions
- Provide guidance on CO2e declaration message types for industry use cases
- Create/recommend technical tools for CO2e declaration message types compliance checks and audits

Existing Embodied Carbon Working Groups, Academics and Others

- Provide best practices/principles for environmental integrity and declarations
- Contribute LCI declaration standards
- Determine CO2e declaration message type fields associated with LCI/LCA work

Regulatory and Governance Oversight

- Leadership and guidance on current regulations
- Provide best practices
- Contribute taxonomic structures related to governance
- Create CO2e declaration message types for compliance and regulatory use cases
- Create/recommend technical tools for CO2e declaration message types compliance checks and audits

Technology Experts/Organizations

- Provide technical guidance advisory for best practices
- Assist in development/use of 3rd party services & tools for creating, compliance checking, passing, parsing and expressing <CarML> data within CO2e declaration message types structure

Technical Design Architects & Ecosystem Thinkers

- Provide meta level thinking about deep design principles for scale and performance associated with:
 - Governance (open source / fit for purpose)
 - Schema & taxonomy
 - Tools & implementations
 - Ease of use
- Lend expertise in <CarML> and CO2e declaration message types systems architecture design and development

Carbon-ML Ecosystem Consortia Governance Framework

An important roles of each consortium is the establishment of a governance framework delineating the roles and responsibilities of each member based on expertise and skillsets. Whereas Carbon-ML.org will act in an oversight and advisory capacity during the governance formation process which will define the control and direction of the creation and development of the standardized CO2e declaration message types among others, it is ultimately the responsibility of the consortium to establishment the governance structure.

A summary example of the potential governance roles and responsibilities of the Carbon-ML ecosystem consortium members, would be as follows:

	Global Standards Organizations	Industry Companies, Associations	Industry Oversight Organizations	Environment Groups, Academics	Regulatory & Government	Technology Experts	Technical Architects
Goals	Guidance, expertise, best practices, use cases	Guidance, expertise, create/design message types	Contribute taxonomies, guidance on message types, recommend technical tools	Best practices, principles on use of standards & integrity of declarations	Leadership & guidance on current regulations, best practices	Technical guidance & expertise, best practices, assist with development decisions	Technical guidance & expertise on architecture & design, best practices
Governance responsibility	Leadership on guiding principles, verification & usage of taxonomies & datasets	Leadership on guiding principles, guidance on schemas, taxonomies, datasets	Guidance on compliance checks, usage of data, assessment of message types	Guidance on compliance checks process & assessment of standards usage	Guidance on message type regulatory fit for purpose, compliance checks, & usage	Leadership on technical development procedures, cybersecurity, & QA process	Leadership on technical architecture & design procedures, cybersecurity, & QA process
Committee Members	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions	Provide expert analysis & assist in process, vote & comment on all decisions
Observers	Monitor & provide comments on process, decisions, & developments	Monitor & provide comments on process, decisions, & developments	Monitor & provide comments on process, decisions, & developments	Monitor & provide comments on process, decisions, & developments	Monitor & provide comments on process, decisions, & developments	Monitor & provide comments on process, decisions, and developments	Monitor & provide comments on process, decisions, & developments

Carbon-ML Ecosystem Consortia Schema and Taxonomies

Definition of schema adapted from Merriam-Webster.com/dictionary/schema, May 2022:
A diagrammatic presentation, broadly a structured framework or plan.

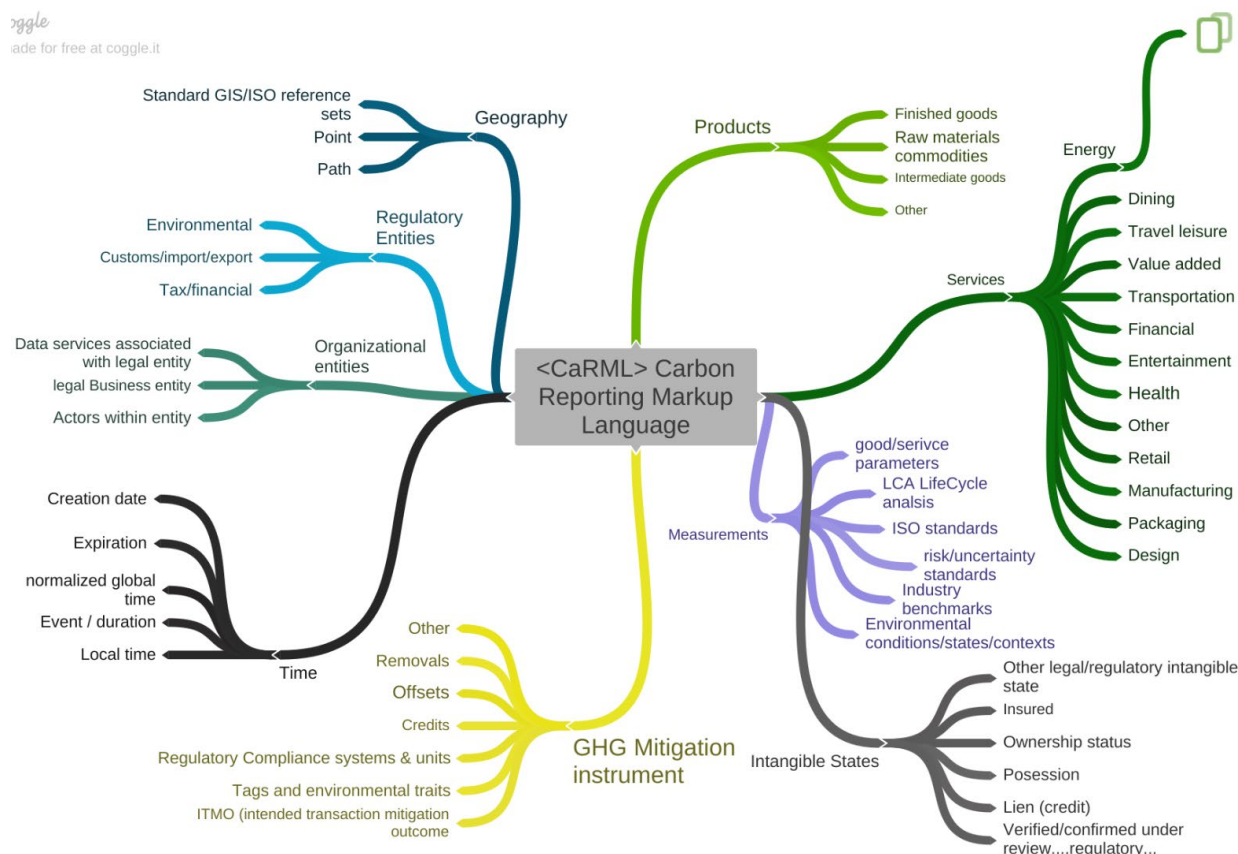
Definition of taxonomy adapted from Merriam-Webster.com/dictionary/taxonomy, May 2022:
The study of the general principles of scientific classification.
Orderly classification of {things} according to their presumed natural relationships.

The Carbon-ML Ecosystem is made of schemas and their taxonomies. Each Carbon-ML Ecosystem Consortium contributes schemas and taxonomies to the overall Carbon-ML Ecosystem.

A Carbon-ML schema is a high-level diagram of its underlying taxonomies. The taxonomies of a schema are a hierarchical classification structure of the groups, sub-group, and individual data points represented with the end stage being a unique ID that references the end-stage data point.

Schemas and related taxonomies can be for product and/or services classifications, CO2e measurement standards such as LCA and LCI, verifier attributes, geolocations, etc.

An early diagrammatic example of what a Carbon-ML Ecosystem Schema may resemble is:



Carbon-ML Ecosystem Schema (Branch) and Taxonomy (Node) Principles:

The Carbon-ML Ecosystem has several **classifications of schemas**, providing information for a **standardized, maintainable context** based CO2e declaration.

Schemas have **taxonomies representing individual entities / products / services / measurements / etc.** that create a unique description at the **branch end node**.

(Example of a schema/taxonomy to be added)

Extensibility and Reference Schema principles:

- Where possible inherit / use existing global standards
- Clarity over completeness, and CO2e declaration defensibility over strict accuracy
- If in doubt leave it to the interface and keep things light
- Utilize existing schemas and taxonomies where possible
- Support multiple technical languages and platforms
- Open-source whenever possible
- **Solution (hacker) Oriented.** If a formal <CarML> message type or taxonomy is not supported, extend it with an informal extension that works, then submit a merge request for the extension. Languages are a do-ocracy! What works and gets used, lives. All else is deprecated or pruned.

Schema related Taxonomy = Intelligent Context:

Repurpose industry, standards/measurements, regulatory, etc. schemas, taxonomies, and data contexts:

- Re-use data from other systems leading to faster integrations and acceptance
- Data should retain value / global context and be exchangeable between external systems
- CO2e data objects with context should be shared to automate publishing / consuming data

Customizable for Regional and Local Variants:

Examples of possible regional / local variants to be reflected in the end branch/schema:

- Context and maintenance can occur at global or local levels to provide static reference, or operating information

Carbon-ML Ecosystem Standard CO2e Declaration Language <CarML>

There is a need for a standardized language for easily understood and shareable CO2e declaration message types.

Carbon-ML.org is developing a standard language for CO2e declarations, called <CarML>.

What is <CarML>

<CarML> is an open-source extensible markup language for CO2e declarations.

<CarML> can be expressed in XML or JSON and can be adapted for other languages.

<CarML> supports multiple extensible schema for structured machine communication about the carbon CO2e associated at a product and/or service level. It enables the structured sharing of CO2e declarations at each point along the supply chain.

- The <CarML> extensible schema is meant to evolve and extend as a framework, not being fully proscriptive of any one technology, solution, and/or interpretation.
- There is no <CarML> schema end solution. <CarML> is extensible for usefulness such that a small part of the tool or tags can be implemented and still be useful.
- <CarML> schemas are shaped by ecosystem stakeholders from existing schemas and taxonomies.

<CarML> provides transparent machine readable CO2e data to accelerate supply chains reporting, creating carbon awareness and enabling mitigation efforts.

In short:

- <CarML> enables shared context for reporting embodied carbon data objects.
- <CarML> is initially open-source XML or JSON standard for how product/service information about CO2e is created, processed, distributed, declared and shared.
- <CarML> messages are essentially CO2e information directly related to a specific product and/or service.

- <CarML> is useful for many product/service handoffs, such as:

	Business	Government	Machine	Consumer	Other
Business	B to B	B to G	B to M	B to C	B to Other
Government	G to B	G to G	G to M	G to C	G to Other
Machine	M to B	M to G	M to M	M to C	M to Other
Consumer	C to B	C to G	C to M	C to C	C to Other
Other	Other to B	Other to G	Other to M	Other to C	Other to Other

Carbon-ML Ecosystem CO2e <CarML> Message Type

What is Meant by a <CarML> Message Type?

<CarML> messages are standardized to show embodied carbon of a product or service at a point along the supply chain is reported.

<CarML> message types are customized for who is using, receiving, and/or acting-on the CO2e information contained of the message.

A <CarML> Message Type:

- Can be customized by the user(s) and adapted informally for regional norms/system conventions
- Can use any data declarative convention (current Proof of Concept demonstrates XML and JSON)

Complete or partial declarations are allowed using only a single fact or a whole statement of embodied carbon within the product or service.

Compliance and verification of input type, string check for integrity is a dependency built into the object generating and object consuming endpoints/systems of the <CarML> Message Type.

Assigning <CarML> compliant attributes to an object/process can be accessed and verified – as an input string check of the <CarML> Message Type to separate well formed signal from unstructured noise.

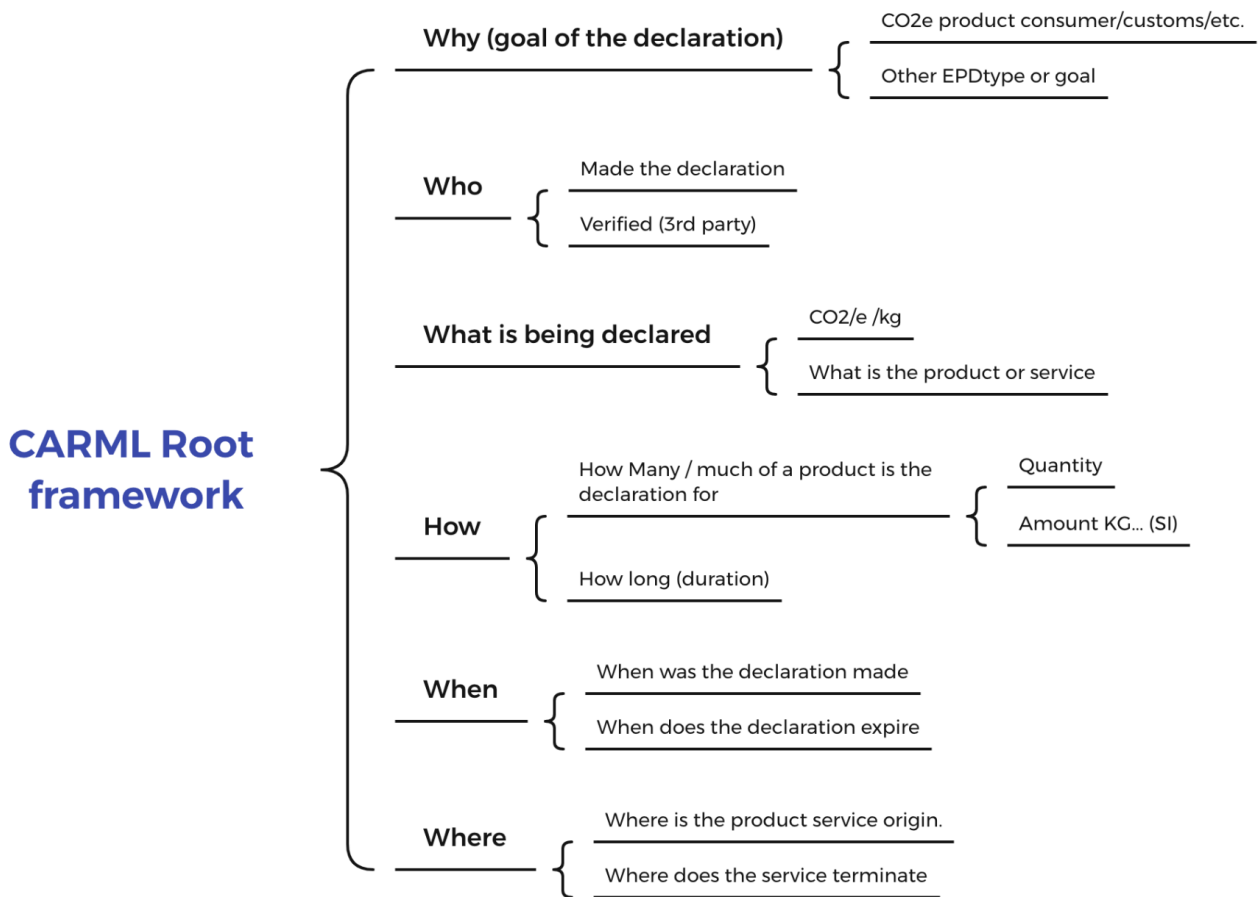
This is structured data, global in context, and drives the point of “how to declare carbon” with the syntax constructed so the <CarML> Message Type can be mapped for further compliance checks and completeness for the user.

Examples of <CarML> message types may include: wholesale, manufacturing or retail required reporting formats based on specific industries; customs or regulatory required information; carbon credits RFQs; or just LCI data - the list is endless.

<CarML> Message Type Root Framework

A <CarML> Message is built by constructing some or all of the <CarML> elements.

These elements are the why, who, what, how, when, and where; subject to data consumer requirements.



<CarML> PoC Message Type General

An example of a <CarML> Proof of Concept (PoC) Message Type General user display for constructing the message type follows.

The interface inputs can be automated or manual with message being expressed in JSON, XML, or other formats.

Table of Contents

General

Unique ID

Amount

Declared Trait Data Source

Declared Trait Measurement Methodology

Location

Verifying Entity

Carbon Credit

PoC Message Type - General

<CarML> Standard for CO2e Declaration

Fields can be customized based on message type, product and/or service.

General

Name

LEGO Star Wars Set

Is Verified

Carbon Credit

Unique ID

Unique ID Source

GS1

Type

Bar Code

Id

67341934C267

Amount

Declared Trait

CO2e

Amount

0.015

Units

KG

Declared Trait Data Source

Declared Trait Data Source

CarbonSig

Declared Trait Measurement Methodology

Declared Trait Measurement Method

LCI

Origin

EU

Methodology

Toy 12345

Date

03/08/2022 12:50 pm

Calculation

Cradle to gate

Location

Location Type

Location at point in time of declaration

Address


7190 Billund, Dinamarca

Long

9.1124

Lat

55.7284



Verifying Entity

Declared Trait Verifying Entity

Reuters

Origin

UK

Company

The LEGO Group 12345

Credentialed Expiration Date

01/01/2023 12:59 am

ISO Auditor

Carbon Credit

Vintage Year

2018

Id

123456789

Carbon Offset Amount

0.015

CANCEL

SEND

<CarML> Message Type Declaration Input Descriptions

The <CarML> Message Type formats/templates developed by the Carbon-ML Consortia will contain version and issuer information, i.e. <CarML> Message Type General v. 1.0. issued by Carbon-ML,

This helps the message consumer know where/what to look for in terms of compliance checking and/or deconstruction/consumption.

Referencing the <CarML> Message Type data classifications as displayed in the <CarML> Message Type PoC General interface, following are the <CarML> Message Type data classification descriptions:

General (Name)

The name of the specific product and/or service.

Why (To Be Added) message intent

Why the CO2e declaration or statement is made and/or updated.

The Why is a process description regarding why the event, declaration of measurement and/or change in measurement of CO2e happened or is being made. It is message intent as understood by sender at time of publishing.

Why was the CO2e for a product and/or service added, altered, updated, or changed among others. There may be a set number of “whys” which can be determined by the Carbon-ML Consortia and may be unique to each <CarML> message type template.

The Why may include non-carbon add events that directly correspond to the CO2e declaration and may have caused a restatement and/or format revision. Examples would include legal state changes/assignments, logical changes/assignments, specifically a duty paid, package certified, audit completed, auditor verified, etc.

These can include changes in the legal or informational state/context of a good or service but involve no physical change or ownership change that would include a transfer of environmental rights and claims in a system of record. These also include re-statement for regimes which may use different CO2e representational conventions.

What (Unique ID)

What the CO2e declaration references. What product and/or service is being referenced and identified.

Example reference and identification schema could include:

- GS1 Barcodes
- ISO codes / industry conventions for objects
- SIC or standard descriptions of service definitions which may have regulatory, legal or industry standard descriptors analogous to SIC codes down to 9 digits
- Life Cycle Assessment (LCA) reference data
- 3rd party publicly available schema or taxonomy for industry

The <CarML> Message Type includes fields to input the Source identification schema, taxonomy, or other; the type of identification; and the unique ID reference for the product and/or service.

Amount (Amount)

The amount of the declared trait most often has two or more facts such as a value and a unit label which may be defined by the creator of the <CarML> Message Type. There may be a drop-down to ease selection. The Carbon-ML consortia may determine certain standardizations, for example CO2e expressed in kg associated with the product and/or service. The data fields may also include whether this amount is an absolute measurement, used reference data, or used a combination of primary and reference data.

Who (Declared Trait Data Source)

Basically, Who made the CO2e declaration. Elements can include who owned, measured, declared, referenced, verified etc. the CO2e in a product and/or service at this point along the supply chain. The Who would comprise any entity within the Carbon-ML ecosystem with a defined role. An entity can be a company, individual, governance organization, reference data set, etc. Examples of Carbon-ML ecosystem entities and related roles for context include among others:

Who would declare, measure, reference CO2e:

- Corporations
 - Commodity producers
 - Service providers
 - Other value added
- Governments
- Regulators
- NGO's and IGO's
- Researchers
- Verifiers such as auditors
- Consumers

Ecosystem Roles:

- Owners of a product or service
- Consumer of a product or service
- Procurement agents
- Taxonomy or Schema maintainer/provider
- Carbon registrar or issuer
- 3rd party Verifier or reviewer
- Product innovators
- Individual viewer or observer
- Technical developer
- Regulatory system
- Procurement system

How (Declared Trait Measurement Methodology)

How was this fact about CO2e assessed, derived, etc. What was the measurement system and specific methodology within that system that was used. What was the resultant CO2e amount. How were the boundary conditions of the LCI determined. How was the quality of data determined (primary, reference etc.)

When (Date and Time)

When did the CO₂e measurement occur, when was it declared. The When involves a point in time event such as an ISO standard time convention likely linked to UTC. Time can be expressed as relative elapsed time such as relative to the location (where) something occurred or as an absolute reference to an event.

Examples of When (Time context of declaration):

- Process, start, end, completion
- Product or service event
- Time of system and data entry update
- Product or service or process expiration
- Service initiation
- Service completion
- Time start of process / time end of process (for travel planes etc. mapped to UTC)

Where (Location)

Where did the activity, the CO₂e declaration event, occur. The Where can be a point or service route, a point along the supply chain, etc. The Where can be identified through the use of GIS/ISO standards for maps and geo locations or polygons.

Verifier (Verifying Entity)

Who verified the CO₂e measurement declaration. The verifying entity can be outside auditors, internal department, etc. Information can include the verifying entity name, credentials, methodology used for verification, when credentials expire, etc.

Environmental Mitigation Instruments (Carbon Credit)

Environmental mitigation instruments such as Carbon Credits utilized as instruments supporting claims of embodied carbon reductions.

Environmental mitigation instruments directly assigned to the product and/or service and used as direct offsets can also be included on the form. For example, for carbon credits, in addition to the carbon offset amount, descriptors and identifiers related to the carbon credit such as type, compliance/voluntary, quality, vintage, IREC registry, state such as retired or pending, future, and owner can also be included.

Other instruments or claims supporting embodied carbon data declarations may be regulatory or private sector attestations such as GO's, REC's, SRECs, IRECs, C-capsule, quality declarations, etc.

<CarML> Message Types Use Case Examples

Summary Use cases for users/consumers of <CaRML> messages

- **Purchasing/procurement managers** looking to ingest and compare product or services on carbon quality for buy clean biases or compliance issues to reduce scope 1 & 2 emissions
- **Infrastructure managers** seeking to inventory initial CO2e and ongoing management and maintenance of CO2e
- **Procurement agents** (RFPs (Request for Proposal))
- **Product or service innovators** looking to create declared or lower carbon products and/or services
- **Consumers** and consumer-facing web front-ends tracking and consuming narratives about carbon for specific audiences
- **Customs and border agents** looking to assess carbon border policies
- **Cities, industrial processes** looking to track carbon in goods and services
- **State, National regulators** wishing to ingest carbon related data quickly and easily around products and/or services
- **Governments** National, Regional & local regulatory, customs and border agencies looking to track provenance, NDC's etc. of embodied carbon in products and/or services
- **CSR (Corporate Social Responsibility) managers** for reporting and utilizing in systems in companies looking to simplify data gathering
- **Researchers/NGOs** performing analysis on carbon in products and/or services

Producers/publishers of <CaRML> message types can include:

- Manufacturer/producers & service providers
- Govt. Regulators / agencies
- NGOs
- Verifiers, accounting firms etc.
- Commodity producers
- Customs agents
- Researchers

<CarML> Message Type Category Examples in Support of Use Cases

<CarML> Message Types can be customized for categories, industries, products, services, regulatory usage, risk assessments, other usages, etc.

The Carbon-ML Ecosystem Consortia develop and recognize compliant <CarML> message type categories and related specific message types within each message type category of value to the consortium and related participants.

Each <CarML> message type, has data classifications and recommended fields for inclusion, but not required:

- For each defined category of <CarML> message types, the data fields to be included under the main classifications (Name, Why, What, Amount, Who, How, When, Where, Verified, Environmental Mitigation Instrument) is to be determined by the respective Carbon-ML Ecosystem Consortium.
- The individual Carbon-ML Ecosystem Consortia define appropriate message type categories and message types within those categories, relevant to their industry/product/service and supply chain markets/users served.
- Each <CarML> Message Type Category is customized for the Consortium's requirements and/or individual business or local variations, for example. In addition, other documents can be linked to each message type, as needed, for detailed informational purposes.

For any product or service, multiple message types can be generated simultaneously, as needed, for different purposes.

For example, a manufacturing item may require

- a general message type for supply chain partners,
- a regulatory message type for customs requirements,
- and a carbon credit RFQ (request for quote) to make the item net zero.

These message types can all be generated at the same time from the <CarML> platform.

Some examples of <CarML> Message Type Categories, possible use cases they support, and how they may be customized include:

Consumer Goods (example)

Example use case:

<CarML> Message Type supporting GS1 would allow for "labeling" approximately 100 million consumer products and/or services out of the box and be supported many inventory/supply chain systems.

- Processing managers can compare based on carbon quality
- Consumers can make more informed decisions

<CarML> Message Type - Consumer Goods CO2e Declaration:

Name: The name of the product.

Why: Why the CO2e declaration is being made. For consumer goods, the declaration may be for informational purposes only.

What: The unique ID associated with the product. This may be several fields, basically a hierarchy structure of Category, Name, Unique ID such as a GS1 Barcode, and the Declared Unit.

Amount of CO2e: The amount of CO2e expressed in kg associated with the unit of the product. The data fields may also include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This may be who signed-off on a CarbonSig report or other, an outside service provider, a defined internal department, etc. Also when the CO2e declaration expires.

How: How is the measurement/formula used for the determination of the CO2e. This may be an LCA analysis, internal measurement, from an outside service provider, etc. Fields should represent the methodology with an appropriate number of data fields to impart an understanding of the calculation. Fields may include the Methodology name, types of inputs, type of calculation, etc.

When: When was the CO2e measurement conducted. This is a general date/time field for when the CO2e measurement was first declared which may be from the prior year. Also, when the CO2e declaration expires.

Where: Where was the CO2e measurement conducted. The where may also be the location of the product and a general geolocator input ISO map address and/or longitude and latitude can be used.

Verified: Some CO2e declarations may require verification by third parties. If so, then these fields would refer to who verified the CO2e declaration and some type of identification fields for the verifier, such as a business name, license/unique ID, location, etc.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. If so, these fields would pertain to the carbon credits purchased, unique IDs, and some characteristics about the environmental mitigation instruments such as vintage year, project, location, etc.

Manufacturing (example)

Example use case:

<CarML> Message Types supporting manufacturing would provide assessments of the embodied carbon in products during and manufacturing process and across the supply chain.

- Procurement managers can compare based on carbon quality
- Product innovators can create lower carbon products

<CarML> Message Type Manufacturing:

Name: The name of the item.

Why: The why refers to why the CO2e declaration is being made, and for manufacturing there may be a requirement from purchasers, industry requirements, regulators, etc. The why may consist of several standardized fields to select the reason why the CO2e declaration is being made.

What: The unique item ID. This may be several fields, basically a hierarchy structure of Category, Name, Unique ID such an industry standard classification and/or internal ID system, and the Declared Unit.

Amount: The amount of CO2e expressed in kg associated with the unit of the item. The data fields may include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This may be who signed-off on a CarbonSig report or other, an outside service provider, a defined internal department, etc. Also, when the CO2e declaration expires.

How: How is the measurement/formula used for the determination of the CO2e. This may be an LCA analysis, internal measurement, from an outside service provider, etc. The fields should be representative of the methodology and there should be an appropriate number of data fields to impart an understanding of the calculation. Fields may include: Methodology name, publisher, types of inputs, type of calculation, primary or reference data used, etc. There also may be regulatory requirements in terms of the information required regarding the CO2e measurement and how it was conducted.

When: When was the CO2e measurement declared for the product. This is a general date/time field.

Where: Where was the CO2e measurement conducted. The where may also be the location of the item and a general geolocator input such as longitude and latitude can be used.

Verified: Some CO2e declarations are 3rd party verified. These fields refer to who verified the CO2e declaration and verifier identification fields, such as a business name, license/unique ID, location, etc.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. If so, these fields show the carbon credits purchased, unique IDs, and characteristics about the environmental mitigation instruments such as vintage year, project, location, etc.

Regulatory Requirements (example)

Example use case:

<CarML> Message Type supporting Government and State regulators' understanding of carbon related data, as standardized data allows for better comparability and tracking of embodied carbon within products and/or services.

- Better assessment for procurement processes and service provider selection
- Better assessment of legislation, regulations, and enforcement

<CarML> Message Type supporting international trade/customs and border agents, policies such as the EU Carbon Border Adjustment Mechanism (CBAM)

- More accurate and standardized assessment of reporting of embodied carbon in products and/or services in line with customs carbon border policies

<CarML> Message Type Regulatory Requirements:

Name: The name of the product/item/service.

Why: The why refers to why the CO2e declaration is being made, and for the Regulatory message type, the why would be for regulatory purposes and there should be a field for the agency who issued the requirement and the regulation ID/code.

What: The unique ID associated with the product/item/service. This may be several fields, basically a hierarchy structure of Category, Name, Unique ID such as industry standard classification, GS1 barcode, internal ID system, etc., and the Declared Unit.

Amount: The amount of CO2e expressed in kg associated with the unit of the product/item/service. The data fields may also include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This may be who signed-off on a CarbonSig report or other, an outside service provider, a defined internal department, etc. Also, when the CO2e declaration expires. However, these fields may also need to correspond to recognized providers by the regulatory authorities.

How: How is the measurement/formula used for the determination of the CO2e. This may be an LCA analysis, internal measurement, from an outside service provider, etc. The fields should be representative of the methodology and there should be an appropriate number of data fields to impart an understanding of the calculation, and the required fields may be pre-defined by the regulatory requirements and associated agencies.

When: When was the CO2e measurement conducted. This is a general date/time field for when the CO2e measurement was made. Also, when the CO2e declaration expires.

Where: Where was the CO2e measurement conducted. The where may also be the location of the product and a general geolocator input such as longitude and latitude can be used.

Verified: Some CO2e declarations may require verification by third parties and this may be a regulatory requirement. If so, then these fields would refer to who verified the CO2e declaration and some type of identification fields for the verifier, such as a business name, license/unique ID, location, etc.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. If so, these fields would pertain to the carbon credits purchased, unique IDs, and some characteristics about the environmental mitigation instruments such as vintage year, project, location, etc.

Environmental Mitigation Instruments RFQ (such as Carbon Credit Request for Quote) (Example)

Example use case:

<CarML> Message Types supporting environmental mitigation instruments RFQs (these may more streamlined message types with several fields not necessarily needed, but may require inclusion of other fields as defined by the consortia).

- Automated RFQ process in standardized format facilitating purchase
- Ease of use of process for requesting and determining environmental mitigation instruments type, quality, project, etc.

<CarML> Message Type Environmental Mitigation Instruments RFQ:

Name: The name of the product/item/service.

Why: Why the CO2e declaration is being made. May be optional.

What: The product/item/service unique ID. May be several fields, basically a hierarchy structure of Category, Name, Unique ID such as industry standard classification, GS1 barcode, internal ID system, etc., and the Declared Unit.

Amount: The amount of CO2e expressed in kg associated with the unit of the product/item/service. The data fields may also include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This field may not be needed.

How: How is the measurement/formula used for CO2e determination. This field may be optional.

When: When was the CO2e measurement conducted. This field may be optional.

Where: Where was the CO2e measurement conducted. The where may also be the location of the product and a general geolocator input such as longitude and latitude can be used. This may be useful.

Verified: Some CO2e declarations may require verification by third parties. This field may not be needed.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. There may be several fields needed regarding the type of environmental mitigation instrument the product/item/service owner may want to purchase. Fields required may be type of instrument such as carbon credits, vintage year preference, verification preference, amount of purchase, etc. These fields may also be determined by the Carbon Credit broker/dealers as to what is required to supply a quote.

The RFQ can be electronically sent to several broker/dealers to obtain quotes and other details. However, Carbon-ML only provides an RFQ message type, for transactions to be concluded, this occurs directly between two parties and outside of the Carbon-ML Ecosystem.

Financial Markets Environmental Risk (example)

Example use case:

<CarML> Message Types supporting financial markets investment decision making by providing more accurate tracing and tracking, and comparable representations of embodied carbon within product and/or services by companies.

- Provides financial markets with insight to environmental risk based on carbon quality of products and/or services
- Information could help inform an assessment of the environmental risk a company/service organization may be vulnerable to and may also be part of the overall risk assessment of a company/service organization

<CarML> Message Type Environmental Risk:

Name: The name of the product/service and company.

Why: The why refers to why the CO2e declaration is being made, and for this message type it may be needed as part of an overall credit risk assessment for the company/business line where more detailed inputs for individual products and services would be useful. There may also be other required fields regarding any regulatory and/or governmental requirements regarding CO2e or other environmental factors for the product/service/industry etc. that would need to be included.

What: The unique ID associated with the product/item/service. This may be several fields, basically a hierarchy structure of Category, Name, Unique ID such as industry standard classification, GS1 barcode, internal ID system, etc., and the Declared Unit.

Amount: The amount of CO2e expressed in kg associated with the unit of the product/service. The data fields may also include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This may be who signed-off on a CarbonSig report or other, an outside service provider, a defined internal department, etc. Also, when the CO2e declaration expires.

How: How is the measurement/formula used for the determination of the CO2e. This may be an LCA analysis, internal measurement, from an outside service provider, etc. The fields should be representative of the methodology and there should be an appropriate number of data fields to impart an understanding of the calculation. For a credit risk assessment, there may be required fields and/or a more detailed understanding of the CO2e calculation that would need to be included.

When: When was the CO2e measurement conducted. This is a general date/time field for when the CO2e measurement was made.

Where: Where was the CO2e measurement conducted. The where may also be the location of the product and a general geolocator input such as longitude and latitude can be used.

Supply Chain and Usage Considerations: For a credit risk assessment, there may be some requirements regarding the use of the product/service and/or supply chain tracking. Data fields such as supplier and consumer, locations, etc. may be helpful. Any additional data fields can be added to message types as determined by the related Consortium and users of the message types.

Verified: Some CO2e declarations may require verification by third parties. If so, then these fields would refer to who verified the CO2e declaration and some type of identification fields for the verifier, such as a business name, license/unique ID, location, etc.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. If so, these fields would pertain to the carbon credits purchased, unique IDs, and some characteristics about the environmental mitigation instruments such as vintage year, project, location, etc.

Transportation Services (example)

Example use case:

<CarML> Message Types supporting transportation services may provide inputs to a product or another services embodied carbon declaration in a standardized format, and allow for further assessments of carbon quality.

- Measurement for passenger transportation services by air, rail, bus etc.
- Comparison of transport used for shipping by route, shipper, mode etc.

<CarML> Message Type Transportation Services:

Name: The name of the transportation service.

Why: The why refers to why the CO2e declaration is being made, and for transportation services it may be a requirement of the companies using those services, regulatory requirements, informational purposes, etc.

What: The unique ID associated with the service. This may be several fields, basically a hierarchy structure of Category, Name, Unique ID such as industry standard classification and/or some type of internal unique code. This can also be standardized by the Consortium.

Amount: The amount of CO2e expressed in kg associated with the unit of the service. The data fields may also include whether this amount is an absolute measurement, or was reference data or a combination of primary and reference data used.

Who: Who made the CO2e declaration. This may be who signed-off on a CarbonSig report or other, an outside service provider, a defined internal department, etc. Also, when the CO2e declaration expires.

How: How is the measurement/formula used for the determination of the CO2e. This measurement and how it is calculated may be unique for each transport service type. The data fields, inputs, calculations, etc. can all be customized based on transport type as needed. For example, the data fields and measurement calculation for aviation, ocean shipping, railroads, trucks, etc. may be very different. These fields for “how” the measurement was calculated can be standardized as several broad categories and then a drop-down can be accessed or the message type for each transport type can be customized. Regardless, the data fields should be representative of the methodology and there should be an appropriate number of data fields to impart an understanding of the calculation. Fields may include the Methodology name, types of inputs, type of calculation, etc.

When: When was the CO2e measurement conducted. This is a general date/time field for when the CO2e measurement was made. Also, when the CO2e declaration expires.

Where: Where was the CO2e measurement conducted. The where may also be the initial location of the service and a general geolocator input such as longitude and latitude can be used.

Verified: Some CO2e declarations may require verification by third parties. If so, then these fields would refer to who verified the CO2e declaration and some type of identification fields for the verifier, such as a business name, license/unique ID, location, etc.

Environmental Mitigation Instruments: For a product to be labeled as net zero, an environmental mitigation instrument or supporting declaration/attested attribute such as carbon credits, carbon removals, GOs, RECs may be purchased. If so, these fields would pertain to the carbon credits purchased, unique IDs, and some characteristics about the environmental mitigation instruments such as vintage year, project, location, etc.

<CarML> Message Types Formal vs. Informal

<CarML> Message Types discussed above refer to “formal” <CarML> Message Types recognized formally by the Carbon-ML Ecosystem Consortium and included in the <CarML> technical structure and data level references such as dictionaries, data mappings, business uses, etc. maintained by Carbon-ML.org.

The Carbon-ML Ecosystem and <CarML> are open source and publicly accessible. There will be Carbon-ML Ecosystem Consortia, participants, or others developing their own CO2e declaration message types and extensions utilizing <CarML> that may reside outside of the formal recognized structure governed by Carbon-ML.org.

Informal <CarML> Message Types can be independently supported and managed until they become widely adopted and/or “formally” recognized protocols, at which time they are incorporated into the <CarML> technical structure, architecture, and data level dictionaries/mappings maintained by Carbon-ML.org.

Extensibility via informal message extensions is a key **benefit** of the Carbon-ML ecosystem. Informal messages let any actors extend <CarML> to suit their immediate purposes without wait for Consortia adoption. If a message extension is useful, it will likely get rapidly adopted or co-opted.

There may also be CO2e declaration message types using languages other than <CarML>, but which are also recognized by the Carbon-ML Ecosystem and maintained outside of Carbon-ML.org.

An open system can be a federated one as well, these may be incorporated into the Carbon-ML.org structure once the Carbon-ML Ecosystem Consortia formally recognize them.

APPENDIX A: <CarML> Message Type Technical Design

(page intentionally left blank)

<CarML> Message Type Technical Design Summary

Various scenarios were used to develop the <CarML> Message Types technical architecture such as:

1. The roles and responsibilities of each Carbon-ML Ecosystem Consortium in the development of guiding principles and <CarML> Message Types for their respective industry and usage
2. Support of multiple schema and related taxonomies types
3. Differences in <CarML> Message Type content based on who is using it and for what purpose
4. A UI/UX, <CarML> Message Type Generator Interface where input of content can be fully automated from a users dataset, or manual.
5. Adaptable to regional and country norms by adhering to ISO conventions when possible
6. Incorporate regional and industry differences in Label/Field names (context) and recognized as having the same definition (context)
7. Supports multiple data elements, databases, formats, structures, etc. through domain “translation layers”
8. Technology Agnostic throughout the tech stack (front to backend), including how the <CarML> Message Type is expressed (JSON/XML/other)
9. Adaptable and scalable connectivity to multiple external databases including proprietary databases and maintenance of privacy of information in constructing and generating a <CarML> Message Type
10. Maintains privacy of proprietary information based on user requirements
11. Supports Formal <CarML> Message Types and allows for ease Informal <CarML> Message Types development
12. Expressable in multiple data formats such as JSON and XML for compliance checking at the data field level and further verified by the user/consumer of the <CarML> Message Type

The above considerations were incorporated into the technical elements that work together to build/construct and express each <CarML> Message Type, and are explained in more detail in the following sections and the appendices. These are the:

- Front End UI/UX interfaces which constructs, generates and expresses <CarML> Message Types for users
- Domain Layer which houses the business rules/use cases/data definition dictionary in support of data field mappings through API calls and endpoints
- Back End databases

In addition, the <CarML> Message Type Generator interface and generation of the <CarML> Message Type Technology Design, in summary:

- Maintains context through the use of <Key, Value> pairs for the <CarML> Message Type individual data fields

- Includes a domain layer “translation layer” for the use of multiple external databases, which may include schemas and related taxonomies for product/service classifications and data elements related to CO2e measurements, and <CarML> Message Type input field mapping
- Can call on proprietary databases while maintaining confidentiality and privacy defined by user/owner requirements
- Employs best practices around privacy and cybersecurity

<CarML> apps and supporting components, in summary:

- For <CarML>, apps (generally defined as a software program developed to perform a specific function directly for the user or for another application program) that have already been developed include the PoC for the <CarML> **Message Type Generator w/ Manual Interface** and for the <CarML> Unique ID which is discussed in more detail in a separate section. These apps can be accessed through a web interface and as a separate desktop application.
- Other components being developed in support of <CarML> include defined business entities such as <CarML> defined tags for specific message type fields, business use cases which would contain database type constructs, an embedded field type check process, and a compliance check process for the user/consumer of the <CarML> message type to ensure reasonableness of the data/information contained within the message type.

<CarML> Technical Decisions Made and Summary of Thought Process

Role and Responsibility of Carbon-ML.org in the Technical Development of <CarML>

The role and responsibility of Carbon-ML.org in the technical development of <CarML> and <CarML> Message Types is as follows, among others:

- **Design, development and publication of <CarML> data field tags and definitions** for use with the <CarML> Message Type Generator Interface and the generation of a <CarML> Message.
- **Design of <CarML> Message Types** and related data fields as approved by Carbon-ML.org Consortia members.
- **Provide technical design and development guidance** to Carbon-ML Consortia members and recommendations by providing potential technical implementation design and development solutions. Members may utilize at their discretion.
- **Work with technical service consultants and providers to provide user guides, learnings, and other support** so that they may provide their services for the actual implementation of a <CarML> solution.
- **Maintain and house the <CarML> Unique ID database** (described in following sections).
- **Work with Carbon-ML.org Consortia** members on technical issues and guidance as needed.

Clean Architecture

Domain Driven Design

JSON vs. XML

A new markup/tag structure

HTML and Desktop Apps

<CarML> Message Type Generator Engineering Architecture

This section provides details of how the <CarML> message types could be designed and developed. It is divided into sub-sections: Backend databases, <CarML> Unique ID Generator, Frontend User Interface, Clean Architecture Principles for <CarML> Development.

<CarML> Message Type Generator Interface Backend Database Architecture

For the <CarML> Message Types Generator Interface to understand what data elements, pertain to a specific product and/or service referenced in a <CarML> Message Type Form, are needed to correctly populate those data fields contained in the <CarML> Message Type Form, backend databases containing that data need be accessed.

- For example, to generate a <CarML> Message Type, databases/reference tables/data tables containing the Name, Why, What, Amount, Who, How, When, Where, Verifier, Environmental Mitigation Instruments need to be accessed.
- The product and/or service detailed information would be in tables related to the schema and taxonomy containing that product and/or service data. This would be similar for other sections/field types of the <CarML> Message Type.
- As each <CarML> Message Type section/field may access disparate schemas and taxonomies this presents a common architecture/construction issue in that the primary key/unique ID for the product and/service referenced on the <CarML> Message Type form is not present in each of the data taxonomies, and therefore the data cannot be mapped accurately to that product and/or service and consequently the <CarML> Message Type itself.
- The taxonomy databases/tables require a common identifier/primary key to retrieve data that accurately maps to and corresponds to the defined product and/or service referenced on the <CarML> Message Type.
- This unique product and/or service ID is resolved by generating a unique <CarML> ID discussed in following sections.

The following section describes why a unique <CarML> ID is necessary with a simple example.

Very Basic Database Architecture Design example

Most companies have databases which have been developed over time and reside in a single location and reference products and/or services and related attributes pertaining only to that company's local operating context. Communication between these databases is based on unique reference fields known as primary keys.

When new technology, standard languages, refactoring, etc. occurs, the updates can be somewhat straightforward based on the pre-existing "communication" structure between internal and/or previously mapped external feeds.

For <CarML> many of the databases/taxonomies needed to supply the <CarML> Message Type sections of name, why, who, what, how, when, where, etc. that comprise the <CarML> Message Type Form are external (in addition to a company's internal databases related to the product and/or service, external

3rd party databases and other reference databases may be required for a <CarML> Message Type to be generated).

These databases, internal and external, each have their own unique primary key structure. These databases have likely not communicated with each other in the past. Understanding what specific piece of data relates to what specific product and/or service and consequently what field on the Form increases the complexity of the development of <CarML>.

A basic example follows:

Project: Create a Form (very basic example of a Message Type Generator) for the different elements associated with a physical object such as a “Tree” where the “Tree” Type is the “product” and the other fields are descriptors of that “product” (similar to a <CarML> Message Type). The Tree Type would therefore be the primary key/unique “product”.

The Form would contain: the type of the tree, the type of leaves on the tree, the height of the tree, the geo location of the tree

For ease of explanation, we have selected three tree types: an oak tree, a maple tree, and a pine tree, with the designated attribute to be measured as “Height”

In common vernacular we may describe these trees as:

- This is an oak tree. It has green leaves. The leaves are lobed. The tree is 60 feet. It is located in New York.
- This is a maple tree. It has green leaves. The leaves are lobed. The tree is 70 feet. It is located in Colorado.
- This is a pine tree. It has dark green needles. The leaves are acicular. The tree is 70 feet. It is located in Colorado.
- A person can make the logical inference that all sentences pertain to the first sentence referencing the tree’s type/name.

However, for a technical system, it “hears” or receives:

- Oak. Green. Lobed. 60 feet. New York.
- Maple. Green. Lobed. 70 feet. Colorado.
- Pine. Dark green. Acicular. 70 feet. Colorado.
- The technical system “hears” distinct words, but does not assume any connection or context between the words. So, if you ask a computer: What type of leaves do oak trees have? A null value would be returned as there exists no association between each word.

Relational databases were developed to provide this association. For a single company, building a single database where all information is stored internally, they could design a single table with data fields such as:

Table: Tree Elements
Primary Key Field: Tree Type
Field: Type of Leaves
Field: Color of Leaves
Field: Shape of Leaves
Field: Height of Tree
Field: Location of Tree

The data to populate this table would be in a row/column format such as:

Tree Type (Primary Key)	Type of Leaves	Color of Leaves	Shape of Leaves	Height of Tree in Feet	Location of Tree
Oak	Leaf	Green	Lobed	60	New York
Maple	Leaf	Green	Lobed	70	Colorado
Pine	Needle	Dark Green	Acicular	70	Colorado

The primary key field is the only field with unique values which is why it has been designated as the primary key field. If other fields containing non-unique value were chosen, for a given query, duplicate records and/or incorrect associations would be returned.

A “primary key” field needs to be designated to associate the other attribute fields with the correct tree type, and to have the correct information returned when a query is made.

Normally, a database has multiple tables and associations. If built internally the connections between tables and elements exist within the structure and one can develop “new” Forms without adding too much complexity.

<CarML>'s Added Complexity to Database Architecture Design

Using the above example of the three tree types, in developing <CarML> information is sourced from internal company databases/taxonomies and external 3rd party databases/taxonomic structure and reference data.

For example (very generalized for illustrative purposes):

- Tree type sourced from: Tree Classification Organization
- Type of leaves sourced from: Arbor Organization
- Color of leaves sourced from: Agriculture Organization
- Shape of leaves sourced from: Tree Company
- Height of tree sourced from: External Measurement Advisor
- Location of tree sourced from: Geo location information

Accessing/sourcing the above databases would resemble (information for illustrative purposes only)

Tree Type	Genus
Oak	Quercus
Maple	Acer
Pine	Pinus

Leaf Shape Code	Leaf Shapes
L123	Lobed
S476	Serrated
A243	Acicular

Tree Genus	Leaf Type
Quercus	Leaf
Acer	Leaf
Pinus	Needle

Tree Name	Tree Height in Feet
Oak	60
Maple	70
Pine	70

Leaf Color Code	Leaf Colors
123	Green
134	Dark Green
245	Red

Geo Locator
New York
California
Colorado

Reviewing the above, one can make an association between tree_type table and leaf_type table by joining on “genus”, and one can make an association between tree type table and tree height table by

joining on “tree type”; however, for other tables an association that would pertain to our “product” of Tree Type (a primary key/unique value of tree type) does not exist.

In addition, the tables contain information not relevant to the three tree types we initially selected.

Given the example above, the question is: How can we populate our Tree Elements Form for the Oak Tree, for example, with data that correctly pertains to the unique tree type selected (Oak Tree)?

This is a basic representation of the disparate data types and data set complexity faced in developing <CarML>.

<CarML> Message Type Backend Architecture Design Solution

The solution to the complexity of multiple external databases with no known common unique reference data fields for product and/or service ID is to create a unique <CarML> ID as a primary key for a given product and/or service.

The creation/development of a table that **provides a unique ID** for every product and service within the Carbon-ML Ecosystem for which a <CarML> Message Type can be created will be maintained by Carbon-ML.org and can be utilized as a reference table and is a basic mapping/primary key table.

The next question may be: How does this table help with the primary association issue between disparate internal and external databases?

The answer is for companies to use a unique <CarML> ID when they construct additional internal tables comprised of the data elements related to specific products and/or services and use the unique <CarML> ID as the primary key/reference mapping.

It will be from these table(s) newly constructed from internal and external databases and the unique <CarML> ID table that the Frontend calls to fetch data needed to populate a <CarML> Message Type Form.

Each Carbon-ML Ecosystem Consortium can develop tables for use, as required, as long as the tables contain data elements associated with a unique product and/or service necessary for completion of all derivations of a <CarML> Message Type using and/or mapping to the unique <CarML> ID as a primary key.

Development and design of a <CarML> Unique ID Summary

The following provides guidance for the creation of <CarML> unique ID's for products, services, and companies. A <CarML> user can choose to utilize their own internal system or have a unique ID generated through a system Carbon-ML.org provides. <CarML> is designed to be flexible and adaptable in order to support multiple solutions, designs, architectures, etc.

Step One: Development of the Product and/or Service Unique <CarML> ID Mapping Table

For the <CarML> backend database(s) schema, each database/table of data/information to be used to populate the <CarML> Message Type must relate to the defined product and/or service of that message type. In the <CarML> Message Type, all data fields are directly related to the unique product and/or service listed on the message type.

Therefore, the primary key/unique ID for all database(s)/table(s) in the <CarML> backend schema must include the same product and/or service ID for the data/information related to that unique product and/or service ID to correctly populate the <CarML> Message Type.

As a note, the primary key may consist of one or more data fields in order for it to be unique.

For the <CarML> backend schema, the product and/or service ID must be unique and not replicated or contained in any other database/table and associated with another piece of data. There can be no duplicates.

To maintain unique ID's, the standardized mapping table discussed above needs to be developed.

This product and/or service unique ID mapping table is to be verified and agreed to by each Carbon-ML Ecosystem Consortium and will be maintained by Carbon-ML.org.

This table will be fully open source and transparent so that it can be used and accessed by each Carbon-ML Ecosystem Participant.

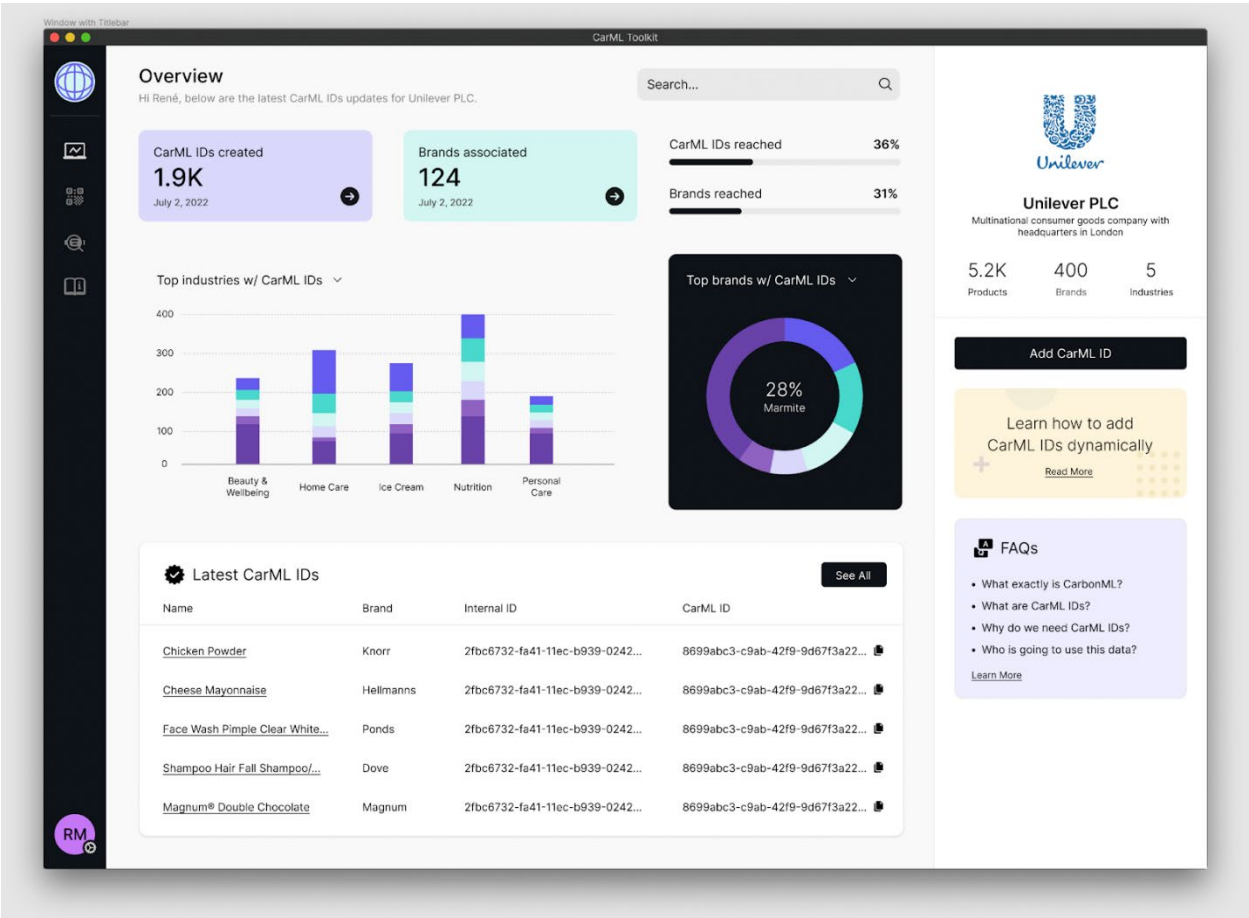
<CarML> Unique ID Generator Toolkit Summary

In addition to providing an unique <CarML> ID generator for products and/or services, Carbon-ML.org has designed a Toolkit with associated Dashboard to display information pertaining to companies registered through the <CarML> site and any products and/or services pertaining to that company that have unique <CarML> IDs.

To ensure accurate mapping of data and display information pertaining to registered companies, Carbon-ML.org is creating a web application to facilitate data synchronization. We have called this app the **<CarML> Toolkit**, to be supported for both web and desktop.

The goal of the <CarML> Toolkit is to accelerate information exchange between Carbon-ML.org and associated companies. Likewise, we this tool to guide and communicate new services, as well as needs required over time.

Take as example the next screenshot:

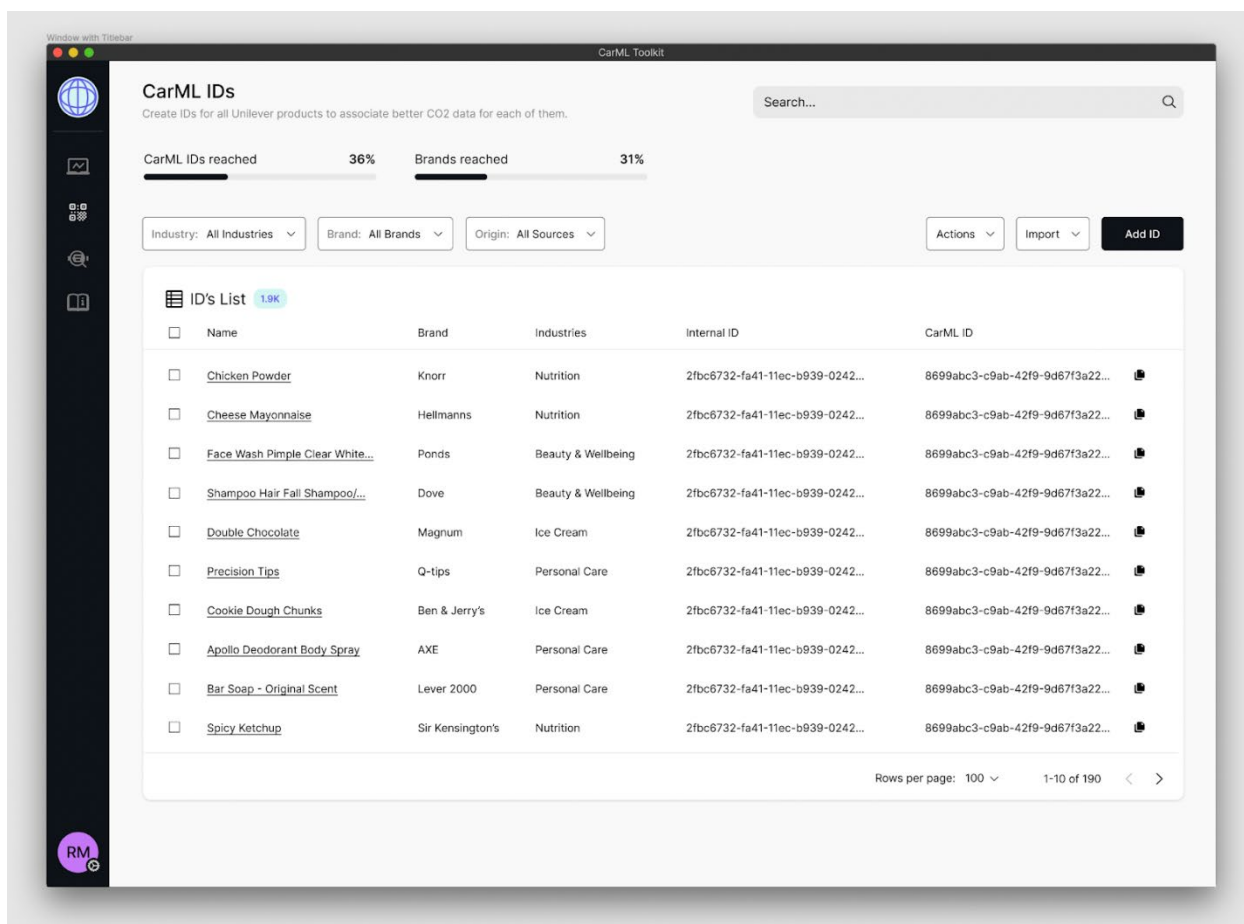


CarML Toolkit - Overview Screen

Companies may have a large amounts of data, but not all products can be included quickly, with unique <CarML> IDs. The <CarML> Toolkit, lets a representative user of a company see which sectors have synced the most products with <CarML> IDs or track the missing items and prioritize.

Carbon-ML.org provides the information, in a convenient dashboard format, that has already been entered by a company for their products and/or services.

In the next screenshot shows the list of unique <CarML> ID's that a company generated for their products and/or services. The screen named CarML IDs lists the unique ID's created for products and/or services associated with the company.



CarML Toolkit - CarML IDs Screen

Access buttons on the Navigation bar at the left side of the application let the user filter by brands, industries or sources if they would like to search for certain products and/or services.

The section ID's List displays summary information about the products and/or services linked to <CarML> IDs previously created, and to copy the <CarML> ID for use with internal and/or external databases.

The user can also open a <CarML> ID that has been created by clicking on one of the multiple links on the list, or can create a new <CarML> ID manually by clicking at the Add <CarML> ID button.

The following screenshot shows the <CarML> ID allowing a user to view and/or create a new <CarML> ID. The user views the new <CarML> ID that has been created and uses this <CarML> ID within their internal and/or externally sourced databases to populate the data fields in a <CarML> Message Type with information directly associated with the product and/or service the <CarML> ID references.

Edit CarML ID
The following identifier was created for Unilever in Jul 16th, 2022.

Product Properties

Name: Chicken Powder

Brand: Knorr Industry: Nutrition

Description: Knorr Chicken Powder elevates the natural flavor and aroma of any dish effortlessly. Its rich meaty taste has made it the trusted chicken seasoning of chefs for brining, marinades, and flavoring.

Internal Id: 630f28c6-fcc2-11ec-b939-0242ac120002

Origin: GSI Version: 0.1 Date: 07/16/2022 03:46 pm

CarML ID Generated
09607064-fcd6-11ec-b939-0242ac120002

Documentation

- Why are CarML IDs needed?
- When do we need to update IDs?
- How can I create IDs automatically?
- What's next?

Latest CarML IDs

Name	Brand	Industries	Internal ID	CarML ID
Chicken Powder	Knorr	Nutrition	630f28c6-fcc2-11ec-b939-0242...	09607064-fcd6-11ec-b939-0242...
Cheese Mayonnaise	Hellmanns	Nutrition	2fbc6732-fa41-11ec-b939-0242...	8699abc3-c9ab-42f9-9d67f3a22...
Face Wash Pimple Clear White...	Ponds	Beauty & Wellbeing	2fbc6732-fa41-11ec-b939-0242...	8699abc3-c9ab-42f9-9d67f3a22...
Shampoo Hair Fall Shampoo/...	Dove	Beauty & Wellbeing	2fbc6732-fa41-11ec-b939-0242...	8699abc3-c9ab-42f9-9d67f3a22...
Double Chocolate	Magnum	Ice Cream	2fbc6732-fa41-11ec-b939-0242...	8699abc3-c9ab-42f9-9d67f3a22...

CarML Toolkit - Edit CarML ID Screen

CarML Toolkit is an effort to build a great communication bridge between CarbonML and companies needing better carbon dioxide declarations.

Company Profile Flow

It's important to note that each company has a profile within <CarML>. Company profiles can be generated by a registered <CarML> user and are publicly displayed and available to all users.

After a user is registered, they can register a company and generate a unique <CarML> Company ID. The <CarML> Toolkit will provide a user guide to explain the steps required for this process.

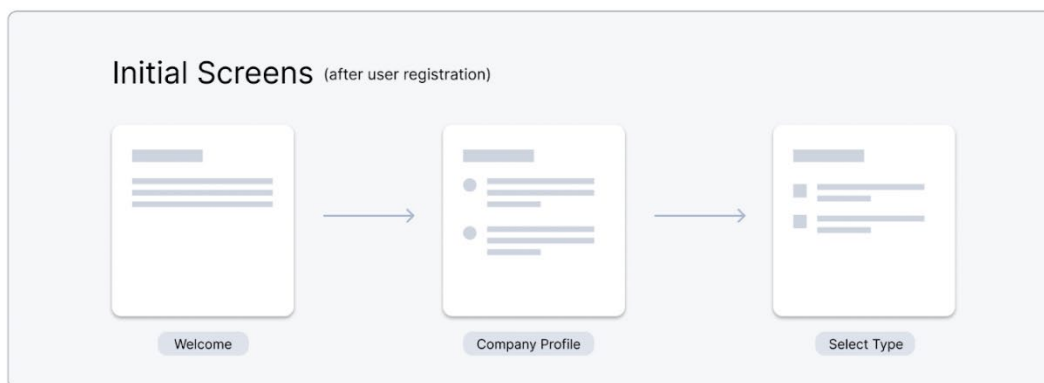
The registered User's ID will be associated with the company record created in order to establish an audit trail. Any registered user can create a company and Carbon-ML.org will maintain the reference table, but will not validate if a user is actually associated with a company. It is the role of the companies within the Carbon-ML Ecosystem to maintain their unique <CarML> Company ID's and associated <CarML> product and/or service ID's internally so they know which ones to use. Carbon-ML.org will

check for duplicate records and normalization of data fields to ensure there is only one record for a distinct product and/or service.

There are 3 steps involved to build a company profile, which is done after a user profile has been created. This information is completely optional but is used if a company would like to track metrics using the <CarML> dashboard:

1. Welcome - A quick introduction to the <CarML> Toolkit
2. Company Profile - Information required to register a Company
3. Select Type - The type of business (products/services) the Company is in

Consider the next graphic:



CarML Toolkit - Company Profile Flow

It is important to note that Carbon-ML.org does adhere to global Privacy Policies and will never share information provided in the steps mentioned below.

The Initial flow, which is subject to revision, is summarized in the following paragraphs.

Welcome Screen

This screen displays information pertaining to the <CarML> Toolkit and unique ID generation:

The <CarML> Toolkit is being provided to guide you through the process of creating unique <CarML> Company ID's, and unique <CarML> ID's for your products and/or services.

The <CarML> Toolkit also provides user guides and utilities to provide a seamless experience and displays information pertaining to companies and/or products and services in a summarized dashboard display.

Company Profile Screen

This screen requests information pertaining to the company to generate the unique <CarML> Company ID:

First, a Company profile is created and a unique <CarML> Company ID is generated, requesting basic information as an example:

1. Name – Company Name
2. Industry – Primary Industry of the Company
3. Headquarters – Headquarters address

Second, some additional information is requested which is optional but may assist in metrics and other audit type functions within <CarML>, for example:

1. What are your main product categories/segments?
2. How many products and/or services do you have?
3. Other information

The <CarML> Toolkit will redirect users to the Overview screen shown in the screenshots above after the company profile and unique <CarML> Company ID is created.

Select Type Screen

This screen is for the generation of the unique <CarML> IDs for products and/or services:

To generate a unique <CarML> ID for a product and/or service please start the process by selecting:

- Products
- Services

Step 2: Mapping of product and/or service CO2e declaration to the corresponding product and/or service unique <CarML> ID

As the <CarML> Message Type Generator Interface is essentially an empty form, and <CarML> essentially a standard by which data items can be represented and communicated, the data needed to populate the <CarML> Message Type Form (such as CO2e declaration details) needs to exist within a backend architecture/structure prior to being imported to and/or accessed.

The Carbon-ML Ecosystem consortia members can create and design <CarML> Message Types as well as other users as both formal and informal <CarML> Message Types can be supported.

Therefore, all database tables/taxonomies that include data needed to populate the <CarML> Message Type for a defined product and/or service, need to contain a unique ID such as the Product and/or Service Unique <CarML> ID (primary key) as established in the mapping table discussed above.

Carbon-ML.org recommends that accessing and communicating with the backend databases be accomplished through a Domain Layer that follows Clean Architecture principles.

Step 3: <CarML> frontend fetching, using pre-defined <CarML> “tags”, the required data for the <CarML> Message Type from the backend databases by going through the <CarML> Domain/Translation Layer which defines business objects/data entities/use cases and essentially “mapping” of the data.

<CarML> Message Type Generator Interface data fields technical flow summary

Once the data fields (general, why, what, amount, who, how, when, where, verifier, environmental financial instruments) and structure for each field (numerical, character, Boolean, etc.) for each unique <CarML> Message Type are determined, defined and approved by the relevant Carbon-ML Consortium, they will be included and supported for use with a <CarML> Domain Layer as use cases/business entities/business rules/etc.

For implementation, it is recommended that a Domain Layer sits between the frontend UI/UX and backend reference databases, and also contains relevant APIs and endpoints to access the correct data that corresponds to the defined data field in the <CarML> Message Type.

This Domain Layer can also be thought of as a de facto “translation layer” for multiple variations of the same data element/business object as defined on the frontend will be “mapped” to a standardized API call / endpoints on the backend. This is done so correct data from the relevant database is selected.

This reduces redundancy, increasing performance and standardization.

The “translation layer” includes automated data field checks to ensure the input is expressed in the correct format. Further detailed compliance checks on the reasonableness of the validity of the data within the <CarML> Message Type will be conducted after the generation of the <CarML> Message Type by the user/consumer prior to sending.

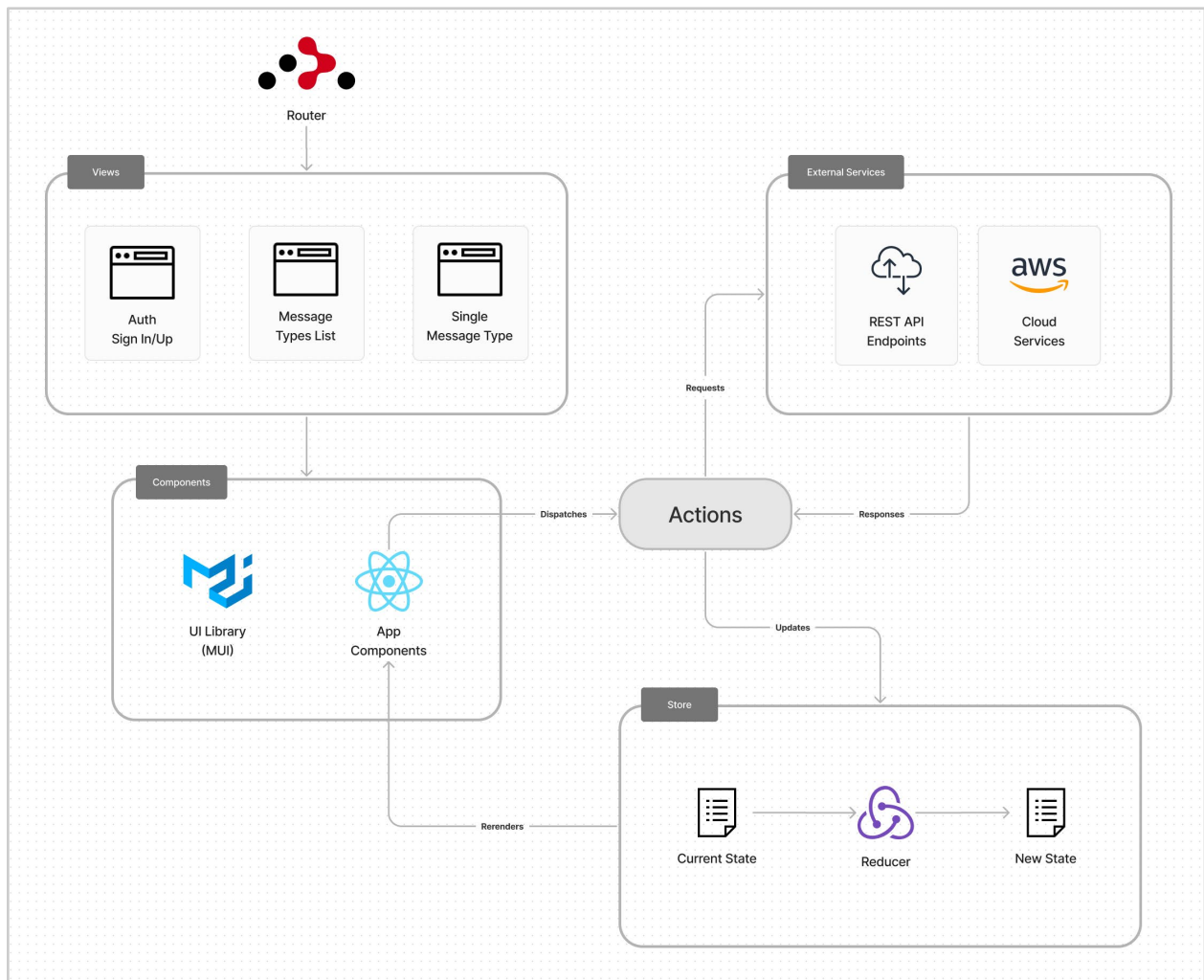
Carbon-ML.org can provide guidance regarding the full technical implementation. For the <CarML> Message Types and defined “tags”, the design is structured to be adaptable for multiple spoken languages; technical languages, data formats and configurations; and be scalable, flexible, and customizable among others.

<CarML> Message Type Generator – The Frontend User Interface Engineering Architecture

Current best practices for Frontend Architecture uses multiple JS (JavaScript)UI libraries UI and synchronizes flow of data between UI and backend databases.

As there will be multiple <CarML> message types created by the Carbon-ML Ecosystem Consortia, our first goal is to create a UI application supporting multiple message types, using cloud based architecture.

The following diagram is the Frontend workflow of the <CarML> Message Types UI generator:



For the <CarML> Message Type generator PoC, we built a UI portal to controls the creation, editing and deletion of <CarML> Message Types.

The <CarML> Message Type PoC is a React app using the [React](#) JS library for building user interfaces, and [Redux](#) along with [Redux Toolkit](#) to control the state of React applications.

Views

The views section represents the UI starting points, the first screens/pages that the user views and interacts with. Technically, these are accessed through routed components commonly known as pages.

- These routed components can act as smart components that can synchronize custom data to be shared across the app, but their main goal in the PoC is to have access to the context of the router and to render static content. This is done through the use of the library [React Router](#).
- For future releases depending on user requirements, the UI routed components can render either smart components or simple components depending on the needs. And, they can also control the data displayed at the UI of the browser by using the library [React Helmet](#).
- For example, the view `MessageTypesList` uses a Store action to fetch all existent Message Types to eventually render them in a table, which is displayed on the UI as a dropdown for user selection. It also updates the browser title to show “Message Types” along with any header metadata we would like to add to that page.

Components

The Components section represents routed smart components and simple components. These components can be part of external libraries or custom libraries created for <CarML> message types.

The simple components are reusable or **presentational components**. They are application agnostic. The PoC uses [MUI](#) (Material UI) to simplify reuse of simple components. In many cases, we extended the components provided by MUI for the PoC and will do so for other <CarML> Message Types to be developed.

The smart components are known as **containers**. Containers are custom components that have access to Stores (stored data, events/actions, and management of the data) allowing them to control the UI in terms of routing, application state including data elements.

Store

The Store section represents information stored with Redux within the React app. Redux uses different concepts for each part of the logic involved:

- **State**, which is any information we want to store such as user information, page displays, data to be accessed, etc.
- **Actions**, to change something in the State
- **Reducer**, to sync a new state given an action change

For the PoC we are currently saving two different types of data in the Store: `UI` and `MessageTypes`.

The `UI` is used to control UI components that exist persistently across the PoC React app, for example the header, sidebar, etc. While `MessageTypes` handles everything related to the specific <CarML> Message Type that has been selected for use.

For the UI, when the user accesses the frontend interface (React app) and performs actions such as selection of message type, data, open/close, etc. (causing a State change in the Store) these actions get re-rendered correctly, as they are controlled by containers (smart components), when there is a change in the Redux Store State. Basically, to access the State in the Store, a selector utility provided by Redux Toolkit is used to make containers (smart components) capable of listening for updates on the Redux Store. Once an update is heard, the smart components get re-rendered automatically reflecting the change that happened in the state.

As an example, a container can trigger actions by using the dispatcher utility provided by Redux. When it is rendered for the first time, it will trigger an action that communicates with an external service. Once it gets the data requested, it proceeds to update the State and re-render the component.

Services

The PoC communicates with external services, like [AWS Cloud Services](#), through the use and configuration of REST APIs. The Services section visualizes when we consume external services in the workflow.

External services are used through asynchronous actions from Redux called thunks. An asynchronous call finishes when a request succeeds or fails. These statuses or the request are managed inside the Store state as well to control the expected UI display.

Summarized Workflow Example

The typical summarized workflow, for example, to fetch the <CarML> Message Types to populate the <CarML> Message Types table on the user interface screen (React app) would be as follows:

1. A user opens the page at the `/message-types` url.
2. The container `MessageTypes` is rendered for the first time.
3. It dispatches the Store action `getMessageTypes()` once.
4. We show a loader component while we wait for data to arrive from the Action.
5. When the data load succeeds from the Action, the Reducer receives the current State and the Action response, and the State at the Store is updated.
6. The containers listening to the Store receive the new State and removes the loader.
7. The `MessageTypesTable` container is re-rendered by receiving the new State.
8. If the new State fails, the loader is removed as well and an Error handler is rendered.

The use of Redux simplifies the process to share data to multiple components that are not related. The use of simple and smart components is a common workflow followed by the React community and is better known as the **Container/Presentational pattern**.

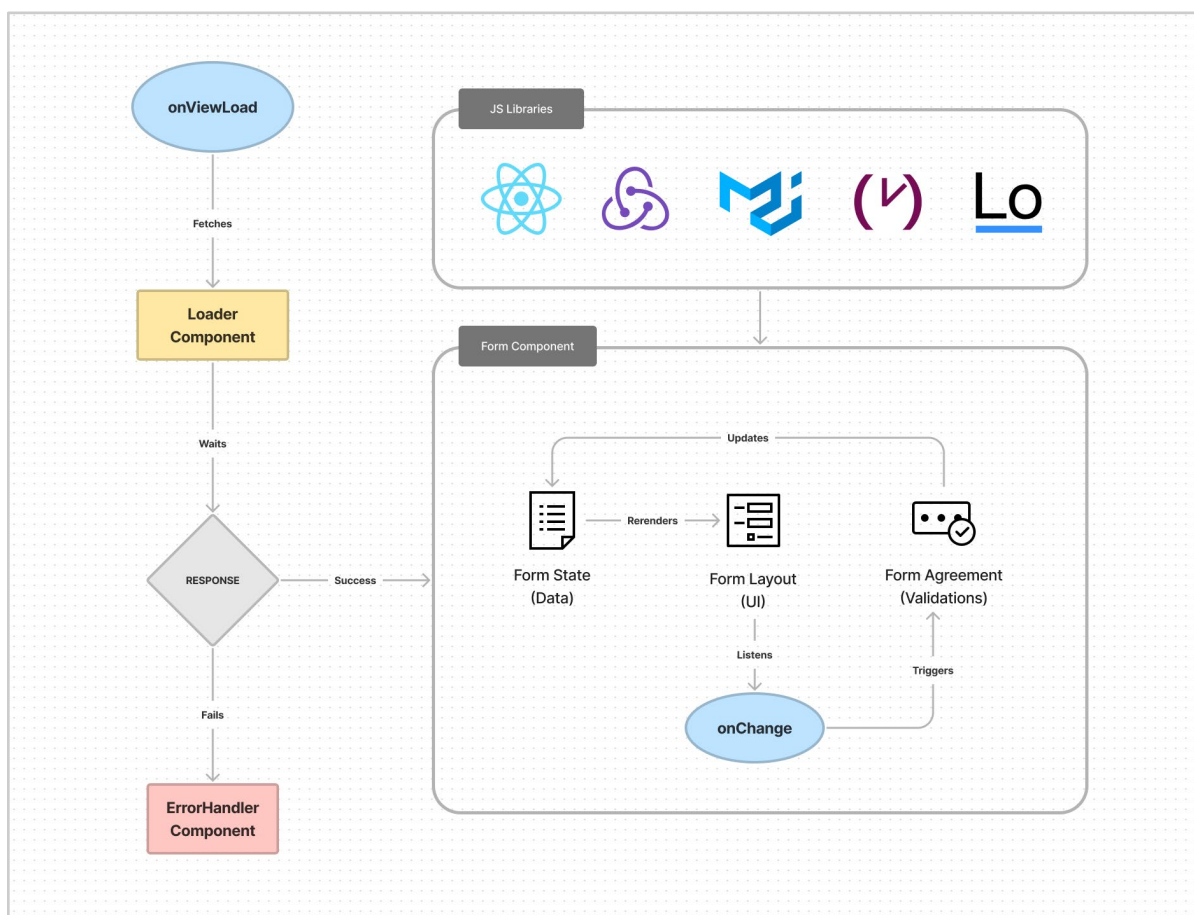
It is important to note that communications with external services can happen when the page loads, on a container load, from user events like a button click, or even from other operations. But the same process can be repeated in those different scenarios safely.

<CarML> PoC Message Type General Form

In general, a form requires multiple processes to happen at the same time. For example, data handling, data synchronization, validations, requests, re-rendering, bindings, multiple custom elements among others. To simplify these processes, third party libraries are being utilized for specific needs for the PoC.

The <CarML> Message Type Form has been developed to demonstrate the feasibility of the <CarML> as a standard and visualize the taxonomy of <CarML> Message Types data, while at the same time providing an opportunity to consider scalability for multiple cases of data schemas for products and services.

The following diagram shows some of the libraries referenced and the common workflow to create the forms relating to each specific <CarML> Message Type.



[React Hook Form](#) was selected because it has incorporated the hooks ecosystem from React very well since its inception. Other form libraries have evolved through years but are still struggling a bit to keep up with the new React ecosystem.

Although React Hook Form is the main library used for the <CarML> Message Type Form, there are more libraries that have been included to assist in handling the data more reliably and efficiently.

External Libraries

Some of the included libraries are used for the purpose of controlling data through the use of JS utilities. [Date-FNS](#), for example, helps us format date strings while [Lodash](#) helps us obtain certain data. MUI is in charge of the UI design with a custom theme and configurations. [Emotion](#) is used to extend MUI components.

The <CarML> Message Type Form handles data rules for field entry through a set of constraints called validations. The [Yup](#) library is being utilized to create agnostic validations schemas that run validations every time the form changes.

To help render useful information about the data displayed on the <CarML> Message Type [x2js](#) is being used to convert JSON to XML and [XMLBeautify](#) is used to format the result to a more readable experience. To print the <CarML> Message Type output the UI [React Syntax Highlighter](#) is used which shows both JSON and XML code inside a non-editable code editor.

Finally, [Maps Javascript API](#) from Google is used to render a map at the custom UI Map field component. This is to help users visualize the geo location of a source.

Form Lifecycle

MUI components were extended to respond to form events. The extended components were text fields, date fields, switches, and others. In addition, custom components were built in order to render displays such as a map.

The <CarML> Message Type Form has its own State and it does not require the data to be saved in the Store. This is handled by React Hook Form by configuring certain properties. However, we still opted to synchronize the data to an `activeMessageType` State in the Store so it can be used outside the Form as well.

All Forms have an initial state. In the case of creating a new <CarML> Message Type, we tell the Form to have an empty object. In the case of updating an existent <CarML> Message Type, the initial State would be the message type data already fetched from the Backend.

React Hook Form expects us to name all fields with the path of the value in demand. So a name might look like `location.geo.lat`, which is related to the specific `lat` key in the JSON data tree.

The library supports registering different methods of data evaluations. This is helpful as we were able to use advanced validators like the Yup library mentioned before.

Since the schema we added for Yup matches the data schema received from the backend, the Form is then able to react when a field changes its value by triggering a validation or any other action we may want to add.

Workflow Example

An example of the common workflow to edit a <CarML> Message Type with existing data through a Form would be as follows:

1. User opens the <CarML> Message Type with url `message-types/5d2656c0-1548-41ff-86a2-36f79ffd75ac`.
2. When the view gets loaded a container dispatches an Action to get the <CarML> Message Type.
3. A loader is shown once the Action is in process.
4. If the <CarML> Message Type succeeds the Form is rendered with its data.
5. The data might use some libraries to be rendered correctly such as dates.
6. When a data field gets updated by the user, the Form updates its State if the validation passes.
7. If the validation fails it responds with errors and the State does not get updated.
8. The cycle is repeated every time an update happens.
9. If the initial fetch fails an Error Handler is rendered.

The lifecycle of the forms depends on multiple factors as the complexity can increase rapidly. To simplify the overall behavior we rely on helpful libraries that reduce the time to develop while providing helpful features to build forms dynamically.

The Form design gives us enough flexibility to create dynamic forms supporting multiple new Message Types that could be rendered from UI schemas. This will be included in subsequent development phases of <CarML>.

Overall <CarML> Message Type Generator PoC Frontend Architecture

The <CarML> Message Type Generator PoC consumes services from a single cloud source so the processes involved in the overall workflow can communicate between each other.

AWS Architecture is utilized where the instances are managed through an AWS Amplify controller. [Amplify](#) is a relatively new Amazon service. Amplify eases the creation of full-stack applications in a short amount of time while providing the opportunity to continue using common AWS cloud services such as [Lambda](#), used for serverless functions, or [Cognito](#) used for advanced user authentication.

There are 4 main processes contained in the Frontend architecture:

- The **web application**, which is the code and assets of the Frontend project.
- The **repository**, where the code, environments and documentation live.
- The **CI/CD**, which involves deployment, hosting, balance and monitoring.
- The **RESTful API**, where integrates multiple cloud services to access and control data.

Web Application

The UI was created with common libraries and services to speed development. The UI is a SPA (Single Page Application) bootstrapped with the [CRA](#) (Create React Application) tool used to start Frontend projects with [React](#) and its ecosystem.

Along with React, the <CarML> Message Type Generator PoC depends on many other libraries as explained in the above section.

The notable tools required on the Web Application are the **Amplify Framework** and the **Amplify CLI**. While the former is used to communicate with the AWS Services at the Frontend level, the latter is used to configure the remote AWS cloud services from our local systems.

The Amplify Framework connects directly to AWS services and provides useful React components that can minimize the effort to build user interfaces that require diverse lifecycle processes. For example, we were able to customize the Sign Up, Sign In and Recovering Password views to follow our own UI/UX designs.

It is worth mentioning that those Auth components mentioned above are persistently and automatically connected with AWS Cognito.

Repository

GitHub is being used to publicly host the <CarML> Message Type PoC code. This is an Open Source project that is visible and can be updated by developers that want to collaborate with it. (Link to be added.)

The repository is configured to have different environments. Currently, it supports `production` and `develop` environment variables so the CI/CD can build and deploy for both versions. It is connected with Amplify to watch for PRs (Pull Requests) that get merged into the environment branches as well. Post POC the repository will include a Staging environment as well for final regression and integration testing prior to production.

Documentation is shared in the repository as well, such as documentation regarding the app, open source collaboration process, templates for PRs, ticket issues and other reports.

CI/CD

With AWS Amplify we were able to connect with our GitHub repository and configure automatic deployments. The administrator tool helps us to visualize the status of the deployments and to run them manually if it's necessary.

The process to build production and development bundles is controlled by the CRA scripts initially installed. Amplify was configured to make it understand where to run each build script. When a deploy finishes, the project can be accessed through an url where the UI is hosted automatically on a CDN powered by [Amazon CloudFront](#).

Another configuration that we added at the CI/CD level is the **notifications process**, which helps us to share the status of our deployments.

Email notifications were supported out of the box, but for Slack notifications a Lambda function was created to connect with [AWS SNS](#) to listen for deployment events and to trigger a custom Slack webhook that is connected with our Slack workspace. (Link to be added.)

RESTful API

This process demands the integration of different AWS cloud services that makes the flow act as a serverless architecture. The way to communicate with the web application is through a single access method, which is via a RESTful API.

By using [AWS API Gateway](#) as an access point we are able to communicate with AWS Lambda functions, which can connect with [AWS DynamoDB](#) and other external databases or services at the same time. The pattern used for endpoints naming is as follows:

REST API Endpoints

Message Types

GET /message-types/	List all message types items
GET /message-types/:id	Get a single message type item
POST /message-types	Create a message type item
PUT /message-types/:id	Update a single message type item
DELETE /message-types/:id	Delete a single message type item

Environment variables are shared at the Lambda functions as well so they can stay synced with AWS Amplify environments. Those variables are useful to connect with external services that share varied contexts.

The Endpoints are currently public but the goal is to create a mix of public and authorized/restricted endpoints depending on the level of security required for some actions.

Clean Architecture Principles for <CarML> Development Recommendation

“The goal of software architecture is to minimize the human resources required to build and maintain the required system.” Robert C. Martin, Clean Architecture

Carbon-ML.org has decided to recommend the guidelines of Clean Architecture with some adaptations as necessary for the <CarML> project and process, in order to architect a technology solution for <CarML> that can among others:

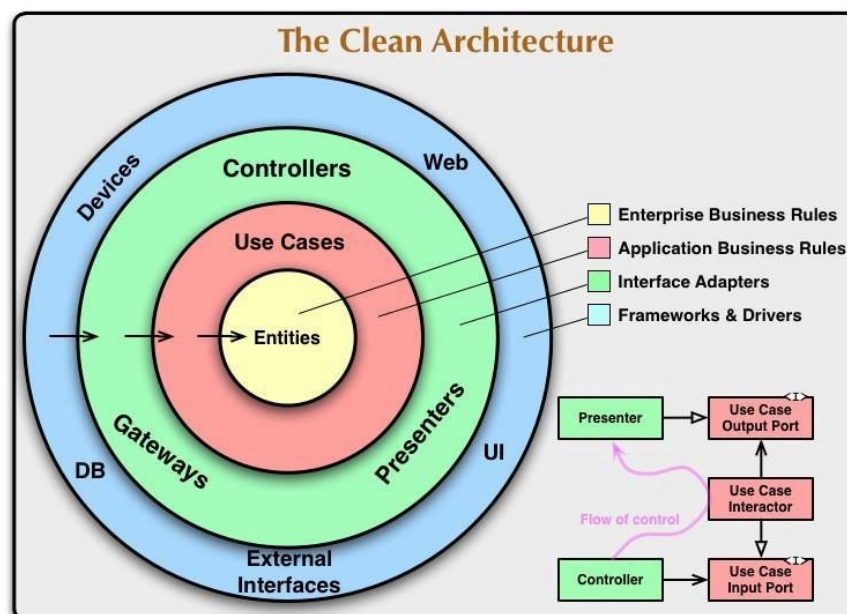
- Provide adaptability and scalability across a wide range of disparate databases,
- Easily incorporate multiple schemas and taxonomies for products and/or services,
- Be technology agnostic,
- Incorporate the business use cases, and business entities in support of the message types required by the Carbon-ML Ecosystem Consortia.

Carbon-ML.org recognizes that each Carbon-ML Ecosystem Consortia member will have their own systems architecture in place; however, to easily incorporate <CarML> the following is one of many approaches/design structures that can be implemented.

A complete use case example is being developed and will be added.

Clean Architecture allows one to keep business logic, or domain logic, together and minimize the dependencies within a system. It follows the concepts of clean code and SOLID principles (discussed further below). Basically, Clean Architecture is designed to create stable apps even when the outer elements such as the UI, databases, or external APIs are constantly changing.

The Clean Architecture is generally represented as follows (Image by Robert C. Martin):



One of the main elements/fundamental part of Clean Architecture is the Domain Layer. Within the Domain Layer, the business and system application rules are developed and control the processing of information independently from the infrastructure and frameworks.

Definitions of Terms:

- The Enterprise Business Rules or Entities refers to the business logic
- The Application Business Rules or Use Cases which refers to the use of the domain knowledge to fulfill specific requests
- Interface Adaptors includes the Controllers, Gateways, Presenters. The Controller is responsible for managing the inputs and outputs of the entire domain, including input checking, and converting domain knowledge to a data structure that is presented on the client side.
- The outermost layer, Frameworks and Drivers, is the external dependencies on the system, including the database(s) and UI.

Some main advantages to using Clean Architecture are that it is testable, maintainable, changeable, easy to develop, easy to deploy, and independent. For example:

- Testable. As each layer has a defined role, test cases can be created for each layer to determine where errors occur within the circle. The business rules can be easily tested without external elements like UI, Database, Web Server, etc.
- Independent of Frameworks: Clean Architecture does not rely on tools from any specific framework and does not use the framework as a dependency in the code. Therefore, the framework can be easily updated or changed with minimal work required to transition.
- Independent of the UI. UI frameworks are on the outermost layer and are therefore just a presenter for data passed from internal layers. The UI can change easily without a corresponding change to the rest of the system.
- Independent of the Database. The application will not need to know which database it is drawing from; therefore, a new database can be adopted with no or minimal changes to the source code. The business rules are not bound to the database.
- Independency of any external agency. The business rules do not have any connection with the external interfaces.

The Dependency Rule for Clean Architecture states that the source code dependencies can only point outwards. Basically, nothing in an inner circle can know anything/be dependent on anything in the outer circle. For Clean Architecture design the domain behavior and use cases should be defined first and then the database.

Example Data flow:

1. The UI issues calls from the Presentation layer
2. The Presentation layer then executes the Use Case

3. The Use Case (Domain layer) will then as the Data layer (access the databases) to send back the result
4. The Data layer will run and send back the result to the Use Case
5. The Use Case layer will incorporate and send back the result to the Presentation layer
6. The UI will display the result

Clean Architecture incorporates SOLID principles, guidelines to write focused, extensible and maintainable clean code.

- **Single Responsibility Principle:** Each module is based on the principle that it should be responsible only for one thing. A module should be responsible to one, and only one, actor.
- **Open-Closed Principle:** A class should be open for extension but closed for modification. Basically, you can add new functionality without changing code.
- **The Liskov Substitution Principle:** Objects in a program should be replaceable by their subtypes, basically objects superclass should be exchangeable with objects of its subclasses, without breaking the application. This requires the objects of the subclasses to behave in the same way as the objects of the superclass.
- **The Interface Segregation Principle:** Many client-specific interfaces are better than one general-purpose interface. Basically, this enables the reduction of side effects as well as the frequency of all required changes by breaking the software up into multiple and independent parts.
- **The Dependency Inversion Principle:** One should depend upon abstractions rather than concretions. High level modules that provide complex logic are reusable and impasse to low level modules which provide utility functions. High level modules do not depend on low level modules, but rather they both depend on abstractions. Abstractions don't depend on details. Details depend on abstractions.

The design of each Clean Architecture Layer, from the inner circle to the outer circle, is described for <CarML> in the following sections.

The <CarML> Clean Architecture Domain Layer

The domain layer of <CarML> is the inner circle. This is where all the enterprise business rules are placed, core-business rules or domain-specific business rules. These business rules should be true and static for <CarML>, any change should be a rare event.

There are three layers to the domain layer: Entities, Use Cases, and Interface Adapters.

- **Entities** can be a collection of data structures and functions necessary for the business rules, or an object with methods that contains business logic within it.
- **Use Cases** is the second layer within the domain layer, they define application specific business rules. Use Cases encapsulate and implement all of the approved Use Cases for the <CarML> app.

They determine the app behavior and control the flow to and from Entities, and can call on Entities to use their enterprise scale rules to achieve the goals of the Use Case.

- **Interface Adapters** are translators between the domain and the infrastructure, the inner and outer circle layers. In this third layer, the software is a group of adapters that convert data to the most convenient format for the Use Cases and Entities. Adapters also format the output from the Entities and Use Cases into a format that is most suitable for external-facing channels. No code of this circle knows anything about the database. **Adapters can be thought of as a “Translator” that converts and relays information in a format that is most usable by the inner and outer layers respectively, and that format can be different for each layer. Basically, this can be thought of as the <CarML> domain “translation/mapping” layer.**

<CarML> Message Type Compliance Check Process

(to be added: diagram, use case, recommended tools and process)

<CarML> Message Type FAQs

(following to be answered)

1. A <CarML> Message Type creator wants to confirm the <Carml> definition and message type of an **element** where? How does one do this?
2. A <CarML> Message Type creator wants to confirm the <Carml> definition and **message type** of a **message** where? How does one do this?
3. A <CarML> Message Type creator wants to check for compliance of an element: Where? How does one do this?
4. A <CarML> Message Type creator want to verify an element example: Product_GS1:1293287389127. What does it return? Does it return a 40 oz. bag of Doritos? What is the lookup or return context?
5. A <CarML> Message Type creator wants to overload a message type with extra elements. Does this occur in the message or after the message? How does one do this using <CarML> tools etc.?
6. How does a User submit an extension, deprecation or new <CarML> Message Type?
7. How does a User see all of the <CarML> Message Types and elements in <CarML>?
8. How does a consortia member or User add a new taxonomy or path by extension?
9. How does a User report that they have found that a database used for verification is old, out of date or broken? What happens. Assume messages are compliant but uncertain.
10. How does a System consuming <CarML> confirm that the version and message types are <CarML> compliant?

Appendix B: Carbon-ML Ecosystem Initial Schemas and Taxonomies Categories and Industries

Categories :

- Oil & Gas
- Retail
- Automotive
- Building & Construction
- Chemicals & Petroleum
- Consumer Goods & Packaging
- Education & Academic Research
- Electronics
- Energy & Utilities
- Finance
- Food & Agriculture
- Healthcare & Life Sciences
- Hydrogen
- Industrial Product or services
- Information Technology
- Metals & Mining
- Transportation

Appendix C: Carbon-ML Ecosystem Definition of Terms (technical and business terms)

(to be defined)

<CarML>

<CarML> version

Carbon Credit

Carbon_ML consortia

Carbon-ML

Carbon-ML Ecosystem

CO₂e

Consortia member

Context

Element

Element compliance

Element dictionary

Element verification

Element version

Environmental Attribute Certificate

EPD Environmental product declaration

Extensible Schema

Formal Message

GHG gases

GitHub

GWP Global Warming Potential

Industry Consortia

Informal Message

LCA Life cycle analysis

LCI Life cycle inventory

Message compliance

Message Generator (tool for creating structure <CarML>) messages could be machine or manual input form

Message types

Message verification

Primary Key

Roles

Schema

Schema version

Taxonomies

Taxonomy contributor/sharer

Types of Schemas/Taxonomies

Verification (element)

Verification (message)

Verification Service

Well-formed Message

Appendix D: Carbon-ML.org Guiding Principles

Carbon-ML combines measuring, tracking and tracing embodied carbon emissions for any product or service with the development of an open-source global ecosystem comprised of extensible schemas from existing taxonomies. We incorporate principles from climate, product or service, sustainability, and technology taxonomies.

1. **Actionable:** Understanding carbon through a common language is a step to making informed decisions. With visibility into embodied carbon in products or services along supply chains, a more sustainable environment and economy may be supported. The Carbon-ML Ecosystem should provide this focus.
2. **Environmental Integrity:** The Carbon-ML ecosystem of <CaRML> language, actors, systems, and declaration should support the highest level of environmental integrity, transparency and acknowledge choices/tradeoffs when these are made such as carbon v. biodiversity under DNH (do no harm) principles.
3. **Goal focused:** The Carbon-ML Ecosystem should have **clearly defined goals, scope, and objectives; be transparent, scalable, robust, and adaptable; be developed with open-source code that is technology neutral, freely available, that maintains a clarity that allows for ease of third-party verification. (Multiple principles)**
4. **Beneficial:** The Carbon-ML Ecosystem should provide value to stakeholders, participants, industry, users, end-product or service consumers, and others in furthering the understanding of embodied carbon in a standardized, easy to understand open-source format.
5. **Open integration:** The Carbon-ML Ecosystem should integrate existing or under development work, using global standards, metrics and methodologies, and product or service and industry taxonomies; and provide flexibility of structure for future integrations and evolution.
6. **Collaborative:** The Carbon-ML Ecosystem evolve through active collaboration,
7. **Iterative & Ongoing:** progressing iteratively with the understanding that usability is prioritized over perfection.
8. **Global access:** The Carbon-ML Ecosystem should incorporate global partners encompassing each of Ecosystem category and general oversight including technical experts, setting defined objectives and responsibilities, tracking measurable actions and contributions, and adding to further development of the Ecosystem, all within the stated scope and governance protocols.
9. **Objective metrics:** The Carbon-ML Ecosystem should be objective with standardized measures and metrics, a measurement and performance indicator reporting system aligned with defined goals and objectives, and a monitoring and management process which includes input from appropriate stakeholders.
10. **Open Governance:** The Carbon-ML Ecosystem should have open and robust governance with consistent policies and definitions; that use global policies and regulations.
11. **Locally relevant & adaptable:** Where possible and not undermining environmental integrity, The Carbon-ML Ecosystem should be adaptable to local, regional, and country based norms and specificities. The declaration of CO2e at each branching point, for each schema and related taxonomy tree, should be adoptable for all regions globally, allowing for regional changes to each resulting branch/stem/leaf.