CGNS Proposal for Extension 0045

Storing Cell-Wise Polynomial Data and Generalising the Description of Curved Grid Elements

Version 3.0

August 4, 2026

CPEX#	0045
Scope	High-order polynomial interpolation for elements and solutions
Champion	Koen Hillewaert
Organization	Cenaero
E-mail	koen.hillewaert@cenaero.be
v3 Revision Author	M. Scot Breitenfeld, The HDF Group — brtnfld@hdfgroup.org
Date First Posted	Mar. 26, 2019
Date of Revision	Aug. 4, 2026
SIDS Status	accepted; awaiting implementation (v3 adopted 2026-08-04)
Filemap Status	accepted; awaiting implementation (v3 adopted 2026-08-04)
MLL Status	accepted; awaiting implementation (v3 adopted 2026-08-04)
Reference impl.	branch `CPEX45_high_order_wip`

Editorial convention. Passages set in the callout shown below reproduce text *verbatim* or *near-verbatim* from CPEX-0045 v2 (K. Hillewaert, Cenaero). Longer technical passages—function-space definitions, pseudo-code listings, and key equations—are marked even when the wording is only lightly adapted from v2. Shorter paraphrased phrases drawn from v2 that are woven into v3 commentary are not individually marked. The surrounding prose (unmarked) is v3 content that formalises, extends, or clarifies the v2 material, and includes the implementation specification adopted in CGNS v5.0.

> **v2 text**
>
> Sample: text in this style is verbatim or near-verbatim from CPEX-0045 v2.

Amendment to Approved Proposal

CPEX-0045 v2 was previously approved by the CGNS Steering Committee. This document (v3) is submitted as a revision to that approved proposal. The committee is asked to evaluate only the *delta* described below. Passages reproduced from v2 are marked throughout with a blue left-rule callout.

What v2 Established

- Two new SIDS node types: `ElementInterpolation_t` and `SolutionInterpolation_t`, both children of `Family_t`.
- `InterpolationType_t` enum with four values: `ParametricLagrange`, `ParametricMonomialsPascal`, `CartesianMonomialsPascal`, `IsoParametric`.
- Support for purely spatial and space-time interpolation via `SpatialDegree` and `TemporalDegree` fields.
- High-order solution data stored in existing `FlowSolution_t` nodes; polynomial orders recorded per solution block.
- Scope restricted to unstructured mesh computations.

Changes from v2 to v3

Each item is labelled **[NEW]** (genuinely new; requires committee vote) or **[CLARIFICATION]** (formalises v2 intent; no semantic change to the approved standard).

SIDS changes:

1. **[NEW] `LagrangeControlPointDistribution_t` enum.** Records the parametric-space distribution of Lagrange control points. Values: `GaussLobattoLegendre`, `Equidistant`, `GaussLegendre`, `WarpAndBlend`. Stored as an optional named child of `ElementInterpolation_t` or `SolutionInterpolation_t`; recommended, but not required, for `ParametricLagrange`. *Rationale:* the attribute is redundant for *interpretation* — the nodal basis is already determined by the stored control-point coordinates together with the element type and order — but naming the distribution lets a reader use a well-conditioned closed-form construction instead of inverting the Vandermonde matrix, regenerate the nodes at full working precision rather than inheriting the writer's `R8` rounding, and lets `cgnscheck` cross-validate the stored coordinates.
2. **[NEW] New `GridLocation_t` enumerator `InterpolationPoints` (value 9), required for all high-order `FlowSolution_t` nodes.** This is a wire-format addition, not a reinterpretation of existing v2 semantics: v2 specified `CellCenter` as the location for high-order solutions (Section 3.4.5). v3 instead designates the new `InterpolationPoints` value as required `GridLocation` for *all* high-order `FlowSolution_t` nodes, both whole-zone (uniform-order) and subset (variable-order via `PointRange`/`PointList`), and for the modal types as well as Lagrange — despite its name, the enumerator labels the per-element block of degrees of freedom, not geometric points (Section 3.1.3). *Rationale:* `CellCenter` mandates a field-array length equal to the number of cells, which contradicts the high-order requirement $\sum_e N_{\mathrm{DOFs}}(e)$ and would fail standard SIDS sizing rules in `cgnscheck`. When the parent `FlowSolution_t` carries a `PointRange` or `PointList` under `InterpolationPoints`, the listed indices are interpreted as *element indices*, and field arrays have length $\sum_e N_{\mathrm{DOFs}}(e)$ summed over the listed elements. For backward compatibility, readers still accept legacy files that use `CellCenter` with an explicit

`PointRange`/`PointList` (Section 6), but conformant *writers* must use `InterpolationPoints` exclusively.

3. **[NEW] `SpatialOrder`/`TemporalOrder` renamed to `SpatialDegree`/`TemporalDegree`.** Renames the two solution-interpolation fields, the `FlowSolution_t` child that carries them (`InterpolationOrders` → `InterpolationDegrees`), and the corresponding accessors (`cg_sol_interpolation_order_{read,write}` → `cg_sol_interpolation_degree_{read,write}`, with scalar arguments `spatialDegree`/`temporalDegree`). *Rationale:* v2 used *order* to mean polynomial *degree* throughout — the common convention calls a degree-p approximation order $p+1$ — so the field name invited exactly the off-by-one misreading Section 3.1 exists to prevent. Renaming removes the ambiguity at its source instead of documenting around it. Element-type *geometric order* (Section 6) is a separate, pre-existing SIDS concept and keeps its name; only the fields CPEX-0045 itself introduces are renamed. No file has ever stored the old names (Section 3.1), so this is a wire-format change with no legacy data to reconcile.
4. **[CLARIFICATION] `InterpolationDegrees` encoding.** Formalises the on-disk representation of `SpatialDegree` and `TemporalDegree` as a 2-element `IndexArray_t` child of `FlowSolution_t`. Information content is identical to v2.
5. **[NEW] Mandatory coordinate normalisation for Cartesian modal, with normative `CharacteristicLength` encoding.** Requires writers to non-dimensionalise the element-local coordinates before populating `MonomialCoefficients`, and to record the factors they used in a normatively-named `CharacteristicLength` `DataArray_t` child of `FlowSolution_t` (Section 5) so readers can recover physical values. Two normalisations are supported and are distinguished by array rank: *isotropic* (rank 1, one factor per element) and *per-axis* (rank 2, `PhysDim` factors per element). Readers use the recorded factors and do not recompute them. *Rationale:* high-order monomials in un-normalised physical coordinates overflow at large or small element sizes, and an isotropic factor alone leaves the modal mass matrix ill conditioned on the high-aspect-ratio cells typical of wall-resolved boundary layers; per-axis scaling is the directional non-dimensionalisation established in the Taylor-basis discontinuous-Galerkin literature [6, 7]. Recording rather than mandating a formula lets a writer dump its native coefficient blocks without rescaling, per the goals of Section 1, and removes any dependence on writer and reader agreeing on a geometric computation. Both encodings and the dedicated MLL helpers `cg_sol_characteristic_length_{read,write}` ship in the v5.0 library on the `CPEX45_high_order_wip` branch.
6. **[CLARIFICATION] `CharacteristicLength` is metadata, not a solution field.** States that a reader enumerating the `DataArray_t` children of a `FlowSolution_t` to build its field list must exclude `CharacteristicLength` by name, and must not apply the field-array length rule to it (Section 6.2). *Rationale:* the node is a `DataArray_t` child of `FlowSolution_t` and so collides with the SIDS convention that such children are fields. A reader that does not exclude it reports a spurious field *and* rejects conformant files, because the metadata length does not equal $\sum_e N_{\mathrm{DOFs}}(e)$. This was a real defect in the reference implementation, invisible on test files with no element sections because field size checking cannot run in that case.
7. **[CLARIFICATION] Element tag stored in `SolutionInterpolation_t` is normalised.** Records that the element type in the node payload is the basic (linear) tag of the family — `TETRA_10` and `TETRA_35` are both stored as `TETRA_4` — which is what makes the uniqueness rule well defined, and restates the lookup accordingly (Section 3.4.4). *Rationale:* v2 and earlier v3 text described an exact-tag match followed by a fallback, implying that high-order tags may be found on disk; they cannot, so the fallback is the operative path for any high-order query.

Resolution behaviour is unchanged.

8. **[CLARIFICATION] Mesh interpolation is nodal only; modal types are solution-only.** States that an `ElementInterpolation_t` node carries no `MonomialCoefficients` child and that its interpolation type is therefore either `ParametricLagrange` or `IsoParametric`. Removes the element-side modal API (`cg_element_monomial_size`, `cg_element_interpolation_coefficients_{read,write}`) and the corresponding File Mapping row (Section 3.4.2). *Rationale:* the third v2 principle states that *the mesh is always defined using interpolations in parametric space, by specifying the set of control points*, and the three-way choice of the second principle is offered for the *solution* interpolation. Earlier v3 text nonetheless allowed `ParametricMonomialsPascal` on the mesh, which contradicted that and would have required changing the element connectivity description. No approved v2 capability is withdrawn.

New normative sections (no equivalent in v2):

- **[NEW] Mid-Level Library API** — complete C and Fortran 2003 read/write interface for all CPEX-0045 node types, including `LagrangeControlPointDistribution` accessors.
- **[NEW] SIDS File Mapping** — exact on-disk array shapes, data types, and DOF-ordering conventions.
- **[NEW] Implementation Specification** — polynomial order limits, validation requirements, error-code semantics, and default values.
- **[CLARIFICATION] Practical order ceiling of the monomial bases** (Section 3.2.4) — informative note that the monomial mass matrix is Hilbert-like, so conditioning degrades exponentially with p regardless of normalisation, limiting `ParametricMonomialsPascal` and `CartesianMonomialsPascal` to roughly $p \leq 2$–3 in double precision. Adds no requirement; it bounds the usable range of an already-approved feature and points at the L^2-orthogonal bases deferred to a future CPEX.

Implementation evidence: The v3 additions are implemented in the CGNS library branch `CPEX45_high_order_wip`: C API (`cgnslib.c/h`), internal readers (`cgns_internals.c`), Fortran bindings (`cgns_f.F90`), and `cgnscheck` validation (including a `-s` "strict CPEX-0045" mode that enforces the order ranges of Section 6). C and Fortran test programs pass. The Cartesian-modal `CharacteristicLength` on-disk node defined in Section 5 is shipped together with the `cg_sol_characteristic_length_{read,write}` helpers in the same branch; the Fortran wrappers follow the standard `_f` suffix convention. Both `CharacteristicLength` encodings, the input validation, and the re-write semantics are covered by the dedicated regression test `src/tests/test_characteristic_length.c`. The bidirectional element-type lookup of Section 3.4.4 is provided by `cg_solution_interpolation_find`.

Deferred to a future CPEX: explicit symbolic representation of individual interpolation functions (noted as a potential extension in v2), which is also the mechanism by which L^2-orthogonal modal bases (tensor-product Legendre, Dubiner/Koornwinder on simplices) would be described — these are what high-order modal methods require beyond the practical order ceiling of the monomial bases specified here (Section 3.2.4); and space-time mesh geometry (e.g. ALE), which would add a temporal-order attribute to `ElementInterpolation_t` or introduce a dedicated space-time mesh-interpolation node.

Motion. The CGNS Steering Committee moves to adopt **CPEX-0045 version 3.0** as an amendment to the previously approved v2, incorporating: (1) the `LagrangeControlPointDistribution_t` enum and its associated child node, **recommended** rather than required (D-01); (2) the `GridLocation_t` enumerator **`InterpolationPoints`**, **retained under its existing name** (D-02), for all high-order solutions, both uniform-order (whole-zone) and variable-order (`PointRange`/`PointList` subset); (3) the **rename** of `SpatialOrder`/`TemporalOrder` to `SpatialDegree`/ `TemporalDegree` (D-03), and of the `FlowSolution_t` child that carries them to `InterpolationDegrees`, as the normative on-disk encoding; (4) mandatory coordinate normalisation for Cartesian modal interpolation, with a normative `CharacteristicLength` child of `FlowSolution_t` supporting both an isotropic (rank-1) and a per-axis (rank-2) encoding, recorded by the writer rather than recomputed by the reader; (5) the Mid-Level Library API, SIDS File Mapping, and Implementation Specification sections as normative components of the standard; and (6) the clarifications listed above, namely that `CharacteristicLength` is excluded from the `FlowSolution_t` field list and that the element tag stored in `SolutionInterpolation_t` is the basic (linear) tag. The control-point lattice helper raised alongside D-01/D-02/D-03 is **deferred to a future CPEX**, not adopted here.

Result: ADOPTED — SIDS, Filemap and MLL, by the CGNS Steering Committee on **2026-08-04**. See the meeting minutes at https://github.com/CGNS/CGNS/wiki/2026 (section 2026-08-04) for the roll call and the vote on each decision and ballot item.

Contents

1 Motivation and Scope

v2 text

The aim is to cater for the large diversity of high-order methods by including a specification of the interpolation spaces for solutions in the file, fixing the bare essentials: the coordinate system. The choice of Lagrangian interpolation spaces is applicable to the mesh as well as the solution description. This generalisation will:

- allow accurate representation of data in a range of popular interpolation spaces which could otherwise only be approximated;
- allow use of the native storage of the application for the Lagrangian description, leading furthermore to
  - a simpler, more robust, and generic implementation of drivers;
  - more efficient I/O by straightforwardly dumping data blocks;
  - avoiding loss of precision for very specific point distributions;
- avoid the need to redefine the format for each new interpolation type/order;
- cater for space-time methods as well as for ALE computations by including time in the functional expressions.

Currently it is assumed that high-order intra-element interpolation is *restricted to unstructured mesh computations*, as structured meshes do not offer the possibility to individually list elements per order and moreover do not support curved elements. Both modal and nodal interpolations are supported.

There is clear potential for future extension via an explicit specification of mathematical expressions of the individual interpolation functions. This will increase the flexibility for modal interpolation approaches.

2 Rationale

v2 text

This proposal lifts the limitation of fixing the interpolation functions implicitly by imposing the position of Lagrange control points as proposed for geometric interpolation/curved elements in CPEX 0036 and CPEX 0038. Instead, the description of the functional space and its interpolant is integrated as metadata in the CGNS file in order to allow high flexibility as to the choice of interpolants or even coordinates, an automatic procedure to allow for very high interpolation order, and time-dependent interpolants.

3 Extension of the SIDS

3.1 Conventions

"Order" fields are named "degree" in v3. v2 named the spatial and temporal quantities `SpatialOrder` and `TemporalOrder`, but used *order* to mean the polynomial *degree* of the expansion — not the number of terms, and not the degree plus one. $p = 1$ is linear, $p = 2$ is quadratic; a temporal value of $q = 0$ denotes data constant in time, not linear in time. This differs from the common convention under which a degree-p approximation is called order $p + 1$, so a "second-order" method in that sense corresponds to $p = 1$ here. v2's own text already used the two words for the same thing — it describes the modal choices as "the maximum *degree* of a Pascal polynomial space" while naming the recorded field `Order` — so the mismatch was inherited, not introduced by v3.

CPEX-0045 v3 resolves it by renaming rather than by documenting around it: `SpatialOrder` and `TemporalOrder` are `SpatialDegree` and `TemporalDegree`, and the `FlowSolution_t` child that carries them is `InterpolationDegrees` rather than `InterpolationOrders` (Section 5). The corresponding Mid-Level Library accessors are `cg_sol_interpolation_degree_{read,write}`, and their scalar arguments are named `spatialDegree` and `temporalDegree`. A reader no longer has to be told which convention a numeric field uses; the field name states it.

"Geometric order" is a separate, unrenamed concept. The `ElementType_t` tag numbering — e.g. `TETRA_35` versus `TETRA_4` — is conventionally called the element's *geometric order* in the base SIDS, and that usage predates CPEX-0045 and is outside its scope: this amendment renames only the interpolation-degree fields it itself introduces, not the pre-existing element-type vocabulary. Geometric order also denotes a degree in the sense above (a "4th-order" element is degree 4, per Section 6), so the same order/degree distinction applies to it in prose, even though the identifier itself keeps the word "order".

> **v2 text**
>
> **Modifications to the SIDS:**
>
> - addition of a new section 3.4: *High-order interpolation.*
>
> In the remainder of this section we introduce the different paragraphs (with numbering to be adapted) that should be added to the new section.

> **v2 text**
>
> "The CGNS standard allows the user to specify their own interpolation approach for both elements and solution. The basic principles are:
>
> 1. The element coordinate system per element type is a fixed convention.
> 2. For the solution interpolation per each element type and interpolation order, a separate interpolation block is added which provides one of three choices:
>   - the set of control points for Lagrange interpolants;
>   - the maximum degree of a Pascal polynomial space defined in parametric coordinates;
>   - the maximum degree of a Pascal polynomial space defined in Cartesian (non-parametric) coordinates.

The first two cover parametric interpolation, whereas the last covers modal/Cartesian interpolation.

3. The mesh is always defined using interpolations in parametric space, by specifying the set of control points. The standard will only allow element types defined up to now, in order not to modify the element connectivity description.
4. The interpolation is *not* supposed to be the same for the geometry as for the solution, unless explicitly specified that way.
5. In addition to the spatial coordinates, time can also be used as an independent variable.

The following sections describe how interpolations are specified."

3.1.1 Interpolation Type Enumeration: `InterpolationType_t`

v2 text

"`InterpolationType_t` specifies how the high-order interpolants for the solution are defined. `InterpolationType_t` can take four values:

- **`ParametricLagrange`** corresponds to a Lagrange interpolation based upon a set of specified control point coordinates in parametric space for a standard interpolation space.
- **`ParametricMonomialsPascal`** corresponds to modal interpolation functions in parametric coordinates, based on monomials according to the classical Pascal sets.
- **`CartesianMonomialsPascal`** corresponds to modal interpolation functions in a Cartesian coordinate system, centred on the element and parallel to the main axes. The interpolation functions are based on monomials according to the classical Pascal sets.
- **`IsoParametric`** corresponds to using the same interpolation functions for the solution as for the mesh."

3.1.2 Lagrange Control Point Distribution

The parametric coordinates of the control points are stored explicitly in `LagrangeControlPoints` (Section 6.2). Together with the reference domain, fixed by the element type, and the polynomial space, fixed by the element type and order, those coordinates determine the nodal basis uniquely: the functions λ_i satisfying $\lambda_i(\mathbf{u}_j) = \delta_{ij}$ are the solution of a standard multivariate polynomial interpolation problem — obtainable, for instance, by inverting the generalised Vandermonde matrix — and exist and are unique whenever the stored point set is unisolvent for that space. A reader therefore never has to *assume* a distribution, and recording it is not required in order to interpret the file correctly.

Recording it is nonetheless recommended, for three reasons:

- **Conditioning.** A reader that knows the points are Gauss-Lobatto-Legendre (GLL), Gauss-Legendre or Warp&Blend can build the basis from the closed-form barycentric weights of that family, which stays stable at high order, rather than inverting a Vandermonde matrix whose conditioning degrades rapidly as p grows.
- **Precision.** The stored coordinates are `R8`-rounded, whereas the *interior* nodes of the

`GaussLobattoLegendre`, `GaussLegendre` and `WarpAndBlend` families are irrational at all but the lowest orders. Naming the family lets a reader regenerate the nodes at its own working precision instead of inheriting the writer's rounding.

- **Validation.** `cgnscheck` can compare the stored coordinates against the named family and so flag writer errors — a permuted point list, or a $[0, 1]$-versus-$[-1, 1]$ domain convention — that are otherwise invisible.

Recommended attribute: `LagrangeControlPointDistribution_t`. An enumeration, stored as a labelled child node named `LagrangeControlPointDistribution` of `ElementInterpolation_t` or `SolutionInterpolation_t` when `InterpolationType` is `ParametricLagrange` — the same convention used for `InterpolationType_t` (Section 3.1.1), and not the name-matched `DataArray_t` convention used for the sibling `LagrangeControlPoints` and `MonomialCoefficients` children. As with every CGNS enumeration, `Null` and `UserDefined` members exist for completeness, though in practice the node is simply omitted rather than written as `Null`. The four named values:

- **`GaussLobattoLegendre`** (recommended default): 1D points are roots of $(1 - u^2)\, P_p'(u) = 0$, where P_p is the Legendre polynomial of degree p; includes endpoints $u_0 = -1$, $u_p = +1$. Optimal conditioning and quadrature accuracy.
- **`Equidistant`**: $u_i = -1 + 2i/p$. Simple uniform spacing; discouraged for $p > 7$ due to poor conditioning of the Vandermonde matrix.
- **`GaussLegendre`**: 1D points are roots of $P_p(u) = 0$; does *not* include the endpoints. Rarely used for element interpolation.
- **`WarpAndBlend`** (simplices only): specialised distribution for triangles and tetrahedra ensuring well-conditioned Vandermonde matrices [4].

Multi-dimensional construction. For tensor-product elements (`QUAD`, `HEXA`), the chosen 1D distribution is applied independently in each parametric direction. For simplex elements (`TRI`, `TETRA`), `WarpAndBlend` or Fekete distributions are required for $p > 2$; equidistant points may be used at $p = 2$ but become unstable at higher orders.

Precedence: the coordinates are authoritative. `LagrangeControlPoints` and `LagrangeControlPointDistribution` are deliberately redundant, and the redundancy is resolved in favour of the data: the stored coordinates are normative and the distribution name is advisory metadata. A reader *must* construct a basis that interpolates at the coordinates actually present in `LagrangeControlPoints`. It *may* use the named family to obtain those nodes by the better-conditioned or higher-precision route described above, but only after confirming that the family's nodes agree with the stored coordinates to within its chosen tolerance; if they do not, it *must* fall back to the stored coordinates and *should* warn. It must never silently substitute the named family's nodes for the stored ones.

This mirrors the rule adopted for `CharacteristicLength` (Section 3.3), where writers record the factors they actually used and readers must use the recorded values rather than recomputing a formula. In both cases the recorded data wins over anything derivable, so that reconstruction never depends on writer and reader agreeing on a computation.

Correspondingly, a writer *must not* record a distribution name that disagrees with the coordinates it writes. `cgnscheck` compares the two whenever both are present and reports a mismatch: an error under `cgnscheck -s`, a warning otherwise.

Implementation status: this comparison is specified here but not yet implemented on the `CPEX45_-high_order_wip` branch, which validates the attribute's presence and value but does not check it against the stored coordinates. Implementing the comparison requires node generation for the named families, which the library does not currently contain, so it is either to be implemented or the requirement withdrawn — see the note in Section 6.

Design note: the comparison must be permutation-invariant. Unlike the standard-lattice helper discussed in Section 6, this check does *not* require a canonical per-family node *ordering* — it requires only the 1D root-generation formulas (`GaussLobattoLegendre`, `GaussLegendre`), tensor-producted independently per parametric direction for `QUAD`/`HEXA`, or the `WarpAndBlend` algorithm for simplices, each a self-contained numerical computation rather than a transcription from a figure. The comparison should therefore match the stored coordinates against the generated set as sets — a nearest-neighbour bijection within tolerance, not an index-by-index comparison — since the on-disk traversal order of `LagrangeControlPoints` is a writer convention (Section 6.2) and is not required to coincide with any particular generation order. An index-by-index comparison would therefore falsely flag a correctly distributed but differently ordered file. At $p \leq 2$ several named distributions coincide exactly (e.g. `GaussLobattoLegendre` and `Equidistant` both reduce to $\{-1, 0, 1\}$ at $p = 2$), so the check is inherently non-discriminating at low degree regardless of implementation; this is a property of the mathematics, not a defect in the comparison.

Absent attribute. If `LagrangeControlPointDistribution` is absent, a reader *must* construct the basis from the stored `LagrangeControlPoints` coordinates directly. It must *not* substitute an assumed distribution, and no warning is warranted, because nothing is ambiguous. An implementation *may* try to recognise the stored points as one of the named families in order to take the better-conditioned path described above, but must fall back to the general construction when no family matches within its chosen tolerance.

3.1.3 New `GridLocation_t` Value: `InterpolationPoints`

CPEX-0045 adds a new `GridLocation_t` enumeration value:

- **`InterpolationPoints`** (enum value 9): the `FlowSolution_t` data is stored at the control points (Lagrange) or as coefficients (modal) defined by the associated `SolutionInterpolation_t` node in the family. The `DataArray_t` length for each field is $\sum_e N_{\mathrm{DOFs}}(e)$, summed over the elements covered by the block (all zone elements when no `PointRange`/`PointList` is present, or the listed elements otherwise). $N_{\mathrm{DOFs}}(e)$ depends on the interpolation type and orders associated with element e, so for a `MIXED` zone or a block whose listed elements are heterogeneous the per-element DOF count is not constant. DOFs of the first element are stored contiguously, followed by all DOFs of the second element, and so on.

Interpretation of the name. The enumerator is named for the Lagrange case but applies unchanged to the modal types, where the stored values are expansion coefficients not associated with any point. Read `InterpolationPoints` as labelling *the per-element block of interpolation degrees*

of freedom — whatever those are for the declared interpolation type — not as an assertion that the data lie at geometric locations. The length rule $\sum_e N_{\mathrm{DOFs}}(e)$ and the element-index semantics below apply identically to both; only the meaning of an individual entry differs.

Mandatory for high-order solutions. For all high-order solutions, `GridLocation = InterpolationPoints` is required, whether the block covers the entire zone (uniform order) or a subset of cells specified by `PointRange` or `PointList` (variable order). Reusing `GridLocation = CellCenter` for high-order data is incorrect: `CellCenter` mandates an array of length equal to the number of cells in the location domain, which contradicts the high-order field-array length $\sum_e N_{\mathrm{DOFs}}(e)$ and will cause failures in standard CGNS readers and `cgnscheck`.

Index semantics under `InterpolationPoints`. When a `FlowSolution_t` carrying `GridLocation = InterpolationPoints` also carries a `PointRange` or `PointList`, the listed indices are interpreted as *element indices* (1-based), not point indices. The expected length of each field `DataArray_t` child is the sum of N_{DOFs} over the listed elements (Section 6.2). In the absence of a point set, the block applies to every element of the zone in element order.

3.2 High-Order Parametric Interpolation

v2 text

Scope: The parametric interpolation conventions can be used for specifying both curved elements (thereby superseding the standard conventions) and the solution.

3.2.1 Standard Coordinate Systems for Parametric Interpolation

v2 text

The parametric coordinate system is defined per element following Figure 1.

For space-time computations the coordinate system is extended with one dimension, by the tensor product of the spatial coordinate system with the parameter interval $[-1, 1]$ in parametric time τ. The latter corresponds to the physical time slab $[T_n, T_{n+1}]$.

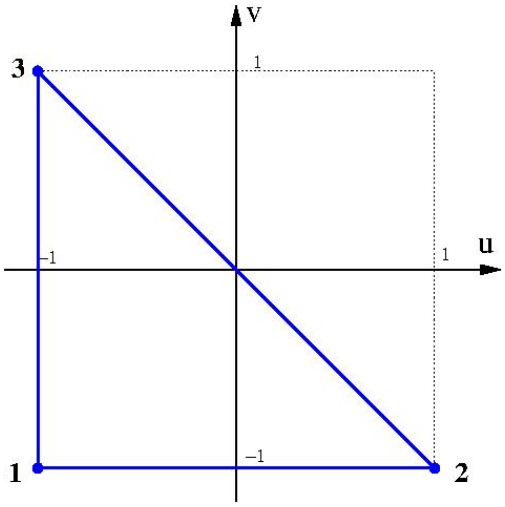
3.2.2 Parametric Interpolation by Specification of Lagrange Control Points

v2 text

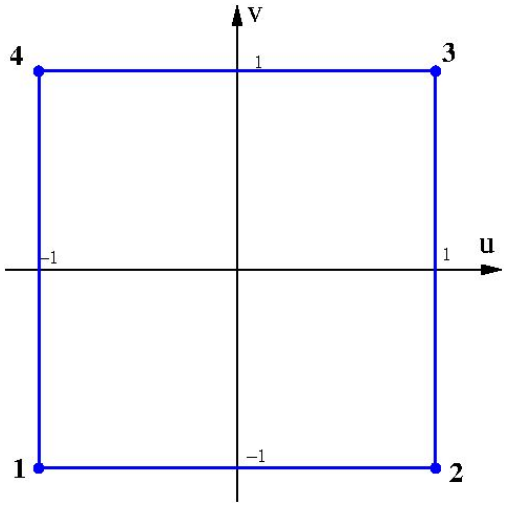
Scope: Both solution and element shape can be specified using this formulation. For elements, this convention allows redefinition of the position and order of the control points with respect to the standard definitions for each `ElementType_t` described in Section 3.2.1. The standard definition continues to be used when no interpolation is explicitly introduced.

Lagrange interpolants require, next to the specification of control point locations, also the specification of the standard function space $\mathcal{V}$. Its cardinality N defines the number of control points that need to be specified.

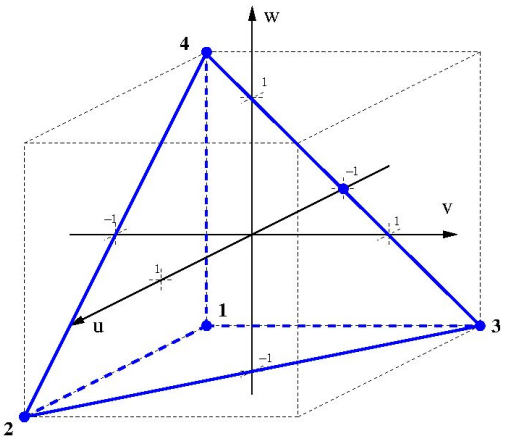
We denote the Lagrange interpolant corresponding to control point $\mathbf{u}_i$ as ${\lambda_i}^1$. Furthermore, we need an arbitrary set of base functions for $\mathcal{V}$, such that $\mathcal{V} = \mathrm{span}(\psi_j,\ j = 0 \dots N-1)$. The



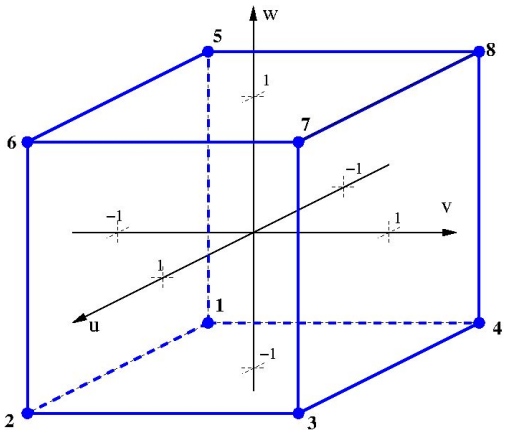
(a) TRI



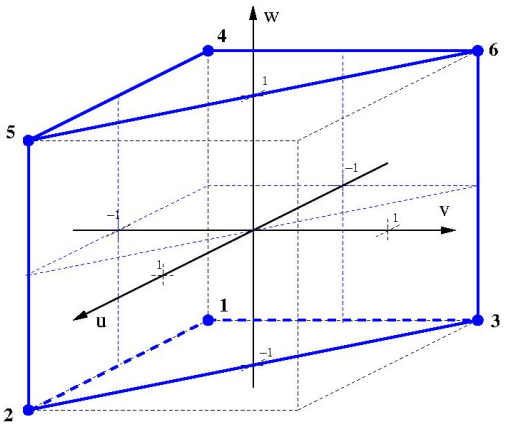
(b) QUAD



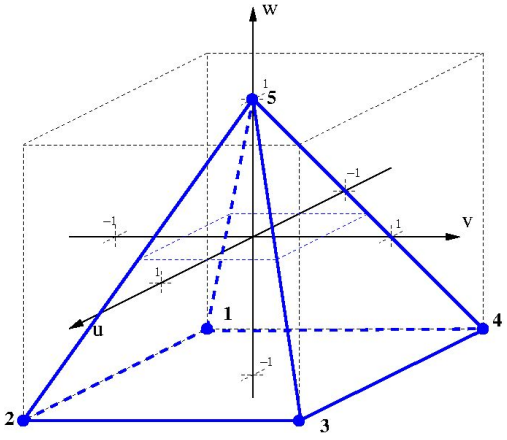
(c) TETRA



(d) HEXA



(e) PENTA



(f) PYRA

Figure 1: Parametric coordinate systems per element type.

Lagrange interpolants are then found as a linear combination of the base functions

$$\lambda_i(\mathbf{u}) = \sum_j V_{ij}^{-1}\, \psi_j(\mathbf{u}) \tag{1}$$

using the inverse of the Vandermonde matrix V associated to the control points $\mathbf{u}_i$ and the basis ψ,

$$V_{ij} = \psi_j(\mathbf{u}_i). \tag{2}$$

The (spatial) parametric function spaces $\mathcal{V}_p(u, v, w)$ for each element type and order p, which support the Lagrange-type interpolation, are listed in Table 1. Next to the standard "complete" element, also incomplete, or so-called serendipity elements are supported in higher dimensions. A first type of serendipity element only specifies control points on the edges. In three-dimensional elements of sufficient order, a second serendipity interpolation can be defined which only excludes control points internal to the element.

v2 text

Element		Function space $\mathcal{V}_p(u, v, w)$		
Type	**Base type**	**Complete**	**Edge Serendipity**	**Face Serendipity**
Line	`BAR_2`	$\mathcal{L}_p(u)$, $N = p + 1$	n/a	n/a
Quad	`QUAD_4`	$\mathcal{Q}^2_p(u, v)$, $N = (p + 1)^2$	$\mathcal{L}_p(u) \otimes \mathcal{L}_1(v) \oplus \mathcal{L}_p(v) \otimes \mathcal{L}_1(u)$, $N = 4p$	n/a
Hexa	`HEXA_8`	$\mathcal{Q}^3_p(u, v, w)$, $N = (p + 1)^3$		
Triangle	`TRI_3`	$\mathcal{P}^2_p(u, v)$, $N = \frac{(p+1)(p+2)}{2}$		n/a
Tetra	`TETRA_4`	$\mathcal{P}^3_p(u, v, w)$, $N = \frac{(p+1)(p+2)(p+3)}{6}$		
Prism	`PENTA_6`	$\mathcal{P}^2_p(u, v) \otimes \mathcal{L}_p(w)$, $N = \frac{(p+1)^2(p+2)}{2}$		
Pyramid	`PYRA_5`	see [3]		

Table 1: Lagrange functional spaces per element and interpolation type. The base type and order p are specified for solution interpolation; the full element type is used to classify element interpolation.

v2 text

In which we use direct sums $\oplus$ and products $\otimes$ of the following standard spaces of order p:

- The linear space: $\mathcal{L}_p(u) = \text{span}\{u^i,\ 0 \le i \le p\}$

[1] Although for 2D and 3D spaces multi-indices are more convenient, we use for simplicity of notation a single compounded index $i = (i, j, k)$. Likewise a compound coordinate $\mathbf{u} = (u, v, w)$ is used.

- Tensor product spaces, $N = (p+1)^d$:

$$\mathcal{Q}_p^2(u,v) = \mathcal{L}_p(u) \otimes \mathcal{L}_p(v) = \text{span}\{u^i v^j,\ 0 \le i,j \le p\}$$
$$\mathcal{Q}_p^3(u,v,w) = \mathcal{L}_p(u) \otimes \mathcal{L}_p(v) \otimes \mathcal{L}_p(w) = \text{span}\{u^i v^j w^k,\ 0 \le i,j,k \le p\}$$

- Pascal triangle/tetrahedron, $N = \frac{(p+1)\cdots(p+d)}{d!}$:

$$\mathcal{P}_p^2(u,v) = \text{span}\{u^i v^j,\ 0 \le i+j \le p\}$$
$$\mathcal{P}_p^3(u,v,w) = \text{span}\{u^i v^j w^k,\ 0 \le i+j+k \le p\}$$

For space-time function spaces (e.g. ALE meshes), the complete functional space is given by

$$\mathcal{V}_{p,q}(u,v,w,t) = \mathcal{V}_p(u,v,w) \otimes \mathcal{L}_q(t) \tag{3}$$

where p and q are the spatial and temporal orders respectively.

Prism (`PENTA_6`) ordering convention. The prism space $\mathcal{P}_p^2(u,v) \otimes \mathcal{L}_p(w)$ is a tensor product of a triangle in (u,v) and a line in w. The implementation requires the line direction w to vary *slowest* when enumerating control points: for each line node along w (w-major), enumerate the $(p+1)(p+2)/2$ triangle nodes in (u,v). This is *not* a simple 3D tensor product like `HEXA`.

Pyramid (`PYRA_5`) cardinality. Following Bergot–Cohen–Duruflé [3], the cardinality of the pyramid functional space at order p is

$$N_{\text{PYRA}}(p) = \frac{(p+1)(p+2)(2p+3)}{6}, \tag{4}$$

giving 5, 14, 30, 55 for $p = 1, 2, 3, 4$ respectively. These are the values returned by `cg_element_lagrange_interpolation_size` and consume the corresponding `ElementType_t` tags `PYRA_5`, `PYRA_14`, `PYRA_30`, `PYRA_55`. The control-point layout convention is the user's responsibility; the library does not provide a built-in pyramid layout helper.

3.2.3 Parametric Modal Interpolation

v2 text

Scope: This type of interpolation only applies to solutions.

v2 text

For solutions, modal interpolation is also allowed. The interpolation functions are based on an ordered set of monomials spanning the Pascal spaces.

The cardinality of the modal basis is the same for every element shape of a given dimension — the binomial coefficient

$$N_{\text{modal}} = \binom{p+d}{d}, \qquad d = \text{element dimension}, \quad p = \text{spatial order, i.e. polynomial degree} \tag{5}$$

For space-time interpolation the cardinality is multiplied by $(q+1)$ where q is the temporal order, again a degree. Consequently a `QUAD_9` element used with modal interpolation of degree $p = 2$ (quadratic; `SpatialDegree` $= 2$, per Section 3.1) stores $\binom{4}{2} = 6$ monomial coefficients (not 9 — 9 is the Lagrange-basis cardinality of the tensor-product space $\mathcal{Q}_p^2$). The helper function `cg_solution_monomial_size` returns N_{modal} for any supported element type and degree combination. Modal interpolation applies to solution data only, never to mesh geometry (Section 3.4.2).

v2 text

According to the underlying dimension the monomials are given by:

- 1D: the monomials $1, u, u^2, u^3, \ldots, u^p$
- 2D — Pascal triangle ordered as:

```
for (int i=0; i<=p; i++)
    for (int j=0; j<=i; j++)
        f[idx++] = u^(i-j) v^j
```

- 3D — Pascal tetrahedron ordered as:

```
for (int i=0; i<=p; i++)
    for (int j=0; j<=i; j++)
        for (int k=0; k<=i-j; k++)
            f[idx++] = u^(i-j-k) v^k w^j
```

The above covers purely spatial interpolation. In case the function space is defined in space-time (eg. for ALE meshes), the complete functional spaces are given by

- 1D in space plus time: the spatial monomials multiplied by monomials in (parametric) time τ, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        f[idx++] = τ^h u^i
```

- 2D in space plus time: the spatial Pascal triangle multiplied by monomials in (parametric) time τ, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        for (int j=0; j<=i; j++)
            f[idx++] = τ^h u^(i-j) v^j
```

- 3D: the spatial Pascal tetrahedron multiplied by monomials in (parametric) time τ, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        for (int j=0; j<=i; j++)
            for (int k=0; k<=i-j; k++)
                f[idx++] = τ^h u^(i-j-k) v^k w^j
```

v2 text

Note that this space-time formulation reverts to purely spatial interpolation — including the ordering — for the (default) temporal order $q = 0$.

3.2.4 Practical Order Ceiling of the Monomial Bases

Both `ParametricMonomialsPascal` and `CartesianMonomialsPascal` use raw monomials. Implementers should be aware that this choice carries an intrinsic conditioning limit that coordinate normalisation does *not* remove. The mass matrix of the monomial set $\{1, x, x^2, \ldots, x^p\}$ on $[-1, 1]$ has entries $M_{mn} \propto \int_{-1}^{1} x^{m+n}\,\mathrm{d}x$, which is a Hilbert-like matrix whose condition number grows exponentially with p. Normalisation (Section 3.3) removes the *scale* disparity between directions and prevents overflow, but it cannot remove this growth.

In practice, modal formulations built on monomials — notably the Taylor-basis discontinuous-Galerkin family [6, 7] — are therefore generally applied at low order, typically $p \leq 2$ or $p \leq 3$ in double precision. Methods that need higher order use L^2-orthogonal bases instead (tensor-product Legendre on quadrilaterals and hexahedra, Dubiner/Koornwinder on simplices), for which the mass matrix is diagonal. This version of the standard has no way to name such a basis; describing one is the province of the explicit symbolic representation of interpolation functions deferred to a future CPEX. Writers requiring high-order modal data should use `ParametricLagrange` with an appropriate control-point distribution (Section 3.1.2) instead.

3.3 Cartesian Modal Interpolation

v2 text

Scope: Cartesian modal interpolation only applies to solutions.

3.3.1 Computation of the Element Coordinate System

v2 text

We specify a local Cartesian coordinate system per element based upon the (simplified) barycenter. Say we note the global coordinates $\mathbf{R} = X\mathbf{e}_x + Y\mathbf{e}_y + Z\mathbf{e}_z$. We proceed by first computing the element barycenter $\mathbf{R}_e$ as the arithmetic mean of the locations of the principal vertices, i.e. the nodes corresponding to those of the associated linear element:

$$\mathbf{R}^e = \frac{1}{N}\sum_{i=1}^{N} \mathbf{R}_i^e \tag{6}$$

The element local coordinates are then defined as $\mathbf{r} = \mathbf{R} - \mathbf{R}^e$; we then use the notation $\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z$.

Correspondingly, an element-local origin of the time dimension is defined as the mid-point of the relevant physical time slab $[T_n, T_{n+1}]$, such that $t = T - (T_n + T_{n+1})/2$.

Numerical stability: mandatory normalization. Using element-local but *unnormalized* coordinates in monomials x^p, y^p, z^p causes catastrophic floating-point overflow or underflow when the physical element size differs greatly from 1 (e.g. kilometres or micrometres), and produces a badly

conditioned element mass matrix when the element is strongly stretched. Conformant writers *must* therefore non-dimensionalise the element-local coordinates before forming `MonomialCoefficients`. Two normalisations are permitted.

Isotropic. A single scale factor per element,

$$\xi = x/h^e, \quad \eta = y/h^e, \quad \zeta = z/h^e. \tag{7}$$

This is appropriate for near-equilateral elements and for general polygonal or polyhedral cells, for which axis-aligned extents are not meaningful.

Per-axis (recommended for stretched elements). One scale factor per coordinate direction,

$$\xi = x/h^e_x, \quad \eta = y/h^e_y, \quad \zeta = z/h^e_z. \tag{8}$$

This is the directional non-dimensionalisation used by the Taylor-basis discontinuous-Galerkin family [6, 7], in which the natural choice is the extent of the element along each axis, $h^e_k = \max_{i,j} |\mathbf{e}_k \cdot (\mathbf{R}^{\text{vertex}}_i - \mathbf{R}^{\text{vertex}}_j)|$. It is what keeps the modal mass matrix well conditioned on high-aspect-ratio cells such as those used in wall-resolved boundary layers: under a single isotropic h^e dominated by the long axis, the short-direction coordinate collapses toward zero and its higher powers lose relative precision, even though no overflow occurs.

The axes remain parallel to the global Cartesian axes in both cases, as required by the definition of `CartesianMonomialsPascal`; only the scaling differs. Rotated or otherwise oriented local frames are *not* permitted, because the modal coefficients of this basis are interpretable as scaled directional derivatives at the element barycentre, and slope limiters in the Taylor-basis family act on them component-wise along the global axes.

Recorded, not recomputed. Writers *must record the scale factors they actually used*, and readers *must use the recorded values* rather than recomputing any geometric formula. The expressions above are recommended defaults, not requirements: a writer whose solver already works in a normalised frame simply records its own factors and dumps its coefficient blocks unchanged, consistent with the goal (Section 1) of using the native storage of the application without loss of precision. Mandating a formula instead of a record would force such a writer to rescale its coefficients on output, and would leave the reconstruction exposed to any disagreement between the writer's and the reader's evaluation of that formula.

On-disk encoding. The scale factors are recorded in a `DataArray_t` child of the parent `FlowSolution_t` node, named `CharacteristicLength`, of `R8` datatype. *The array rank selects the normalisation*:

- **rank 1**, shape $[|\mathcal{E}|]$ — isotropic; one h^e per element.
- **rank 2**, shape $[\texttt{PhysDim}, |\mathcal{E}|]$ — per-axis; `PhysDim` factors per element, with `PhysDim` the fast-varying axis so that the factors of one element are contiguous $(h^1_x, h^1_y, h^1_z,\ h^2_x, h^2_y, h^2_z,\ \ldots)$, matching the `LagrangeControlPoints` layout convention.

$|\mathcal{E}|$ is the number of elements covered by the block: all elements of the zone for a whole-zone block, or the number of elements listed by `PointRange`/`PointList` for a subset block. Element ordering within the array matches the element ordering of the corresponding field arrays (Section 5). All factors

must be strictly positive. The `CharacteristicLength` child is mandatory when the associated `SolutionInterpolation_t` carries `CartesianMonomialsPascal`, and is otherwise omitted.

Implementation status. The node name and both on-disk shapes are normative. The MLL helpers `cg_sol_characteristic_length_read` and `cg_sol_characteristic_length_write` (Section 4.3) carry an explicit `nscale` argument selecting the encoding, and write and read this node directly. `cgnscheck` validates the datatype, the rank, that `nscale` is either 1 or `PhysDim`, and the element count, whenever the node is present; it also reports a missing `CharacteristicLength` on a `CartesianMonomialsPascal` solution (an error under `cgnscheck -s`, a warning otherwise).

3.3.2 Cartesian Modal Interpolants

v2 text

The interpolation space for Cartesian modal interpolations follows the same approach as for parametric modal interpolation and is based on ordered sets of monomials spanning the standard Pascal spaces. Here these are monomials in the element-local Cartesian coordinates x, y, z and (physical) time t, rather than coordinates in parametric space and time. The ordered sets of monomials for a purely spatial interpolation are thus given by:

- 1D: the monomials $1, x, x^2, x^3, \ldots, x^p$
- 2D — Pascal triangle ordered as:

```
for (int i=0; i<=p; i++)
    for (int j=0; j<=i; j++)
        f[idx++] = x^(i-j) y^j
```

- 3D — Pascal tetrahedron ordered as:

```
for (int i=0; i<=p; i++)
    for (int j=0; j<=i; j++)
        for (int k=0; k<=i-j; k++)
            f[idx++] = x^(i-j-k) y^k z^j
```

Again, a space-time interpolation can be obtained by multiplying with temporal monomials, which yields the following sets of (ordered) monomials

- 1D in space plus time: the spatial monomials multiplied by monomials in (parametric) time, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        f[idx++] = t^h x^i
```

- 2D in space plus time: the spatial Pascal triangle multiplied by monomials in (parametric) time, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        for (int j=0; j<=i; j++)
```

```
            f[idx++] = t^h x^(i-j) y^j
```

- 3D: the spatial Pascal tetrahedron multiplied by monomials in (parametric) time, ordered in the following way

```
for (int h=0; h<=q; h++)
    for (int i=0; i<=p; i++)
        for (int j=0; j<=i; j++)
            for (int k=0; k<=i-j; k++)
                f[idx++] = t^h x^(i-j-k) y^k z^j
```

3.4 Overriding the Element Definition and Solution Interpolation

v2 text

Modifications to the SIDS:

- include list of solution/element interpolants in section 12.6 `Family_t`;
- new section 12.10 `ElementInterpolation_t`;
- generalisation of section 7.3 `Elements_t` and example in 7.4;
- new section 12.11 `SolutionInterpolation_t`;
- generalisation of section 7.7 `FlowSolution_t` and example in 7.8;
- renumber sections 12.10 `UserDefinedData_t` and 12.11 `Gravity_t`.

3.4.1 Modification of Section 12.6 (`Family_t`)

v2 text

On a case-by-case basis, i.e. per element type and interpolation order, one can provide alternative mesh and solution interpolants in a dedicated family to the zones in question. This is implemented using dedicated lists of `ElementInterpolation_t` and `SolutionInterpolation_t` leaves within `Family_t`.

```
Family_t :=
```

```
List( Descriptor_t Descriptor1 ...  DescriptorN ); (o)
FamilyBC_t FamilyBC ; (o)
...
List( ElementInterpolation_t ElemInterp1 ...  ElemInterpN); (o)
List( SolutionInterpolation_t SolInterp1 ...  SolInterpN); (o)
};
```

3.4.2 `ElementInterpolation_t` — New SIDS Section 12.10

> **v2 text**
>
> The `ElementInterpolation_t` specifies the geometric interpolation of an element by listing an alternative set of Lagrange high-order control points in parametric space following the element conventions for the coordinate system. In the absence of such a block for a given `ElementType_t`, the standard described in Section 3.2.1 is followed. **It is assumed that the first points correspond to the principal vertices of the corresponding linear element, in the same order (cf. Figure 1).**

```
ElementInterpolation_t :=
```

```
    int Element; (r)    /* ElementType_t */
    DataArray_t<Float,DataSize[]> LagrangeControlPoints; (o)
    LagrangeControlPointDistribution_t LagrangeControlPointDistribution; (o)
    };
```

`Element` is the node's own `I4` payload (the `ElementType_t` value), not a child — the same convention used for `SolutionInterpolation_t`'s triple (Section 3.4.4) and matching the File Mapping in Section 5 and the reader-validation rule of Section 6.3.

`LagrangeControlPointDistribution` may be present when `LagrangeControlPoints` is present (i.e. the interpolation type is `ParametricLagrange`). It is recommended but not required, and carries no information a reader needs in order to interpret the control points; see Section 3.1.2 and the File Mapping in Section 5.

The interpolation type is not stored explicitly as a child node; it is inferred from whether the one optional data array is present:

- `LagrangeControlPoints` present ⇒ `ParametricLagrange`
- absent ⇒ `IsoParametric` (element's own coordinates used)

Mesh interpolation is nodal only. Neither modal type may be attached to an `ElementInterpolation_t` node, and this node therefore has no `MonomialCoefficients` child. This follows the third of the v2 principles quoted in Section 3.1: *the mesh is always defined using interpolations in parametric space, by specifying the set of control points.* The three choices of the second principle — control points, a parametric Pascal space, and a Cartesian Pascal space — are offered for the *solution* interpolation; mesh geometry is restricted to control points, so that the element connectivity description need not change. Accordingly `ParametricMonomialsPascal` and `CartesianMonomialsPascal` are both solution-only, and the only mesh interpolation types are `ParametricLagrange` and `IsoParametric`.

> **v2 text**
>
> **Limitations**: The current proposal maintains `ElementType_t` to describe both element type and geometric order, meaning we cannot go beyond 4th-order interpolation. This choice is motivated by maintaining the `ElementConnectivity_t` leaf in its current shape and the fact

> that currently there is no real need for higher geometric orders.

Uniqueness. At most one `ElementInterpolation_t` per exact `ElementType_t` tag is permitted within a family. Tags differing only in geometric order are considered distinct; for example a single family may carry separate `QUAD_4` and `QUAD_9` `ElementInterpolation` nodes. The constraint is enforced both on write and on read.

Scope — spatial only. `ElementInterpolation_t` describes *purely spatial* mesh interpolation; it carries no `TemporalDegree` attribute. The space-time function-space machinery of Section 3.2.2 therefore applies to `SolutionInterpolation_t` (which does carry a temporal order) but not to mesh geometry under this version of the standard. Space-time mesh geometry (e.g. ALE) is deferred to a future CPEX, which would add a temporal-order attribute to `ElementInterpolation_t` or introduce a dedicated space-time mesh-interpolation node.

3.4.3 Changes in Section 7.3 (`Elements_t`)

> **v2 text**
>
> In case an alternative location for the element control points is specified, the actual elements are defined as before by listing the indices in the coordinate table, with the notable change that the order will correspond to the control point coordinates specified in the corresponding `ElementInterpolation_t` block. If a specific element type is not found among these leaves, the standard convention is applied.

3.4.4 `SolutionInterpolation_t` — New SIDS Section 12.11

> **v2 text**
>
> The interpolation functions associated to the interpolation on a given element type and order are stored in a `SolutionInterpolation_t` leaf attached to the corresponding Family.

```
SolutionInterpolation_t :=

    int[3] InterpolationData; (r)    /* [ElementType, SpatialDegree, TemporalDegree]
    */
    InterpolationType_t InterpolationType; (r)
    DataArray_t<Float,DataSize[]> LagrangeControlPoints; (o)
    DataArray_t<Float,DataSize[]> MonomialCoefficients; (o)
    LagrangeControlPointDistribution_t LagrangeControlPointDistribution; (o)
    };
```

Naming note. v2 referred to this required child informally as `InterpolationName` (see the quoted passage below). The normative on-disk name adopted for v3, consistent with the File Mapping in Section 5 and the reference implementation, is `InterpolationType`. There is only one such child; the two names refer to the same node.

The triple `[ElementType, SpatialDegree, TemporalDegree]` is the node's own `I4` payload (length 3, rank 1), as detailed in Section 5. `TemporalDegree` defaults to 0 when omitted by the writer.

Uniqueness. At most one `SolutionInterpolation_t` per (basic-element-type, `SpatialDegree`, `TemporalDegree`) triplet is permitted within a family.

Stored element tag is the basic tag. The element type in the node's payload is *normalised to the basic (linear) tag of the element family* when the node is written: a writer given `TETRA_10` or `TETRA_35` stores `TETRA_4`. This is what makes the uniqueness rule above well defined — tags differing only in geometric order are not distinguishable after normalisation, so they cannot coexist — and it is what allows one basis description to serve every geometric order of the same family.

Lookup. A query is therefore resolved in two steps: an exact comparison against the stored tag, and, failing that, a second comparison against the basic tag of the queried type. Both must be tried, because the caller may legitimately present either form. A query for (`TETRA_4`, 2, 0) matches on the first step; a query for (`TETRA_10`, 2, 0) or (`TETRA_35`, 2, 0) matches on the second, and resolves to the same node. Note that, because writers normalise, the first step succeeds only for a basic-tag query; the fallback is the operative path for any high-order tag. A triplet matching neither is reported as absent (`CG_NODE_NOT_FOUND`), which is a normal result and not an error. The MLL function `cg_solution_interpolation_find` (Section 4.2) performs both steps, so callers need not reimplement the fallback.

> **v2 text**
>
> The relevant `SolutionInterpolation_t` block will be found using the pair composed by the (basic) element type and interpolation order. The former corresponds to either the actual element tag or, if the corresponding `SolutionInterpolation_t` block is absent, the type of the corresponding linear element. For instance, the interpolation functions for the 2nd-order solution on a 4th-order tetrahedron will be associated to element tag `TETRA_35` or `TETRA_4`. Finally, if `InterpolationName` is not specified, standard interpolation (i.e. constant per element) applies.

The `TETRA_35` alternative in the passage above no longer arises on disk. Because writers normalise the stored tag, the block for that example is always written under `TETRA_4`, and a `SolutionInterpolation_t` tagged `TETRA_35` is never produced. The either/or the passage describes survives only on the *query* side, where a caller may legitimately present the high-order or the basic tag and the lookup resolves both to the same node. The v2 wording predates the normalisation rule stated above and is retained here for the historical record only.

3.4.5 Addition to Section 7.7 (`FlowSolution_t`)

> **v2 text**
>
> In case variable high-order solutions are stored, a separate solution block per interpolation order in space (and time) should be stored. The interpolation orders attached to the zone are indicated by the integers `SpatialOrder` and `TemporalOrder`. The location of the solution is then supposed to be `CellCenter`, and in case of a variable-order solution, one needs to use

> PointRange or PointList to single out the elements which will use the specified order.

```
FlowSolution_t<int CellDimension, int IndexDimension,
    int VertexSize[IndexDimension], int CellSize[IndexDimension]> :=

    List( Descriptor_t Descriptor1 ...  DescriptorN ); (o)
    GridLocation_t GridLocation ; (o/d)
    IndexArray_t<1, 2, int> InterpolationDegrees ; (o/d)
    DataArray_t<R8, 1|2, [NumElements] or [PhysDim,NumElements]>
    CharacteristicLength ; (o)
    Rind_t<IndexDimension> Rind ; (o/d)
    IndexRange<IndexDimension> PointRange ; (o)
    IndexArray<IndexDimension, ListLength[], int> PointList ; (o)
    List( DataArray_t<DataType, IndexDimension, DataSize[]>
    DataArray1 ...  DataArrayN ) ; (o)
    DataClass_t DataClass ; (o)
    DimensionalUnits_t DimensionalUnits ; (o)
    List( UserDefinedData_t UserDefinedData1 ...  UserDefinedDataN ); (o)
    };
```

The IndexArray_t carries [SpatialDegree, TemporalDegree] (length 2, datatype I4). CharacteristicLength is required only for Cartesian modal interpolation; its rank selects the normalisation (rank 1 isotropic, rank 2 per-axis), as defined in Section 3.3.

> **v2 text**
>
> The default value for SpatialOrder depends on the type of interpolation; the default for TemporalOrder is 0.

Implementation note 1 — Accessor functions. The InterpolationDegrees child is read and written through cg_sol_interpolation_degree_{read,write}, which take spatialDegree and temporalDegree as separate scalar arguments while persisting the underlying length-2 IndexArray_t. See Section 5 for the on-disk encoding.

Implementation note 2 — Grid location convention. All high-order FlowSolution_t nodes use GridLocation = InterpolationPoints (see Section 3.1.3), regardless of whether they describe the entire zone or a subset of elements. For variable-order configurations, PointRange or PointList children of the FlowSolution_t list *element* indices, and the per-subset orders are recorded in the InterpolationDegrees child of each FlowSolution_t block. The cgnscheck "strict CPEX-0045" mode flags CellCenter plus InterpolationDegrees as a SIDS conflict.

4 Extensions to the Mid-Level Library

Function signature	Modes
`ierr = cg_element_lagrange_interpolation_size(ElementType_t t, cgsize_t *sz)`	r--
`ierr = cg_solution_lagrange_interpolation_size(ElementType_t t, int os, int ot, cgsize_t *sz)`	r--
`ierr = cg_solution_monomial_size(ElementType_t t, int os, int ot, cgsize_t *sz)`	r--
`ierr = cg_element_lagrange_interpolation_count(fn, bn, fam, ElementType_t t, int *cnt)`	r--
`ierr = cg_solution_lagrange_interpolation_count(fn, bn, fam, ElementType_t t, int os, int ot, int *cnt)`	r--

Table 2: Helper functions: interpolation space sizes and family query counts.

Parameter	Description
`t`	Element type
`os`	Spatial interpolation degree
`ot`	Temporal interpolation degree
`sz`	Output: cardinality of the interpolation space
`cnt`	Output: number of matching interpolation nodes in the family

Table 3: Input/output parameters for helper functions.

4.1 Helper Functions

The `*_size` functions return the cardinality of the Lagrange or monomial interpolation space for a given element type and orders. The `*_count` functions query a family to count how many `ElementInterpolation_t` or `SolutionInterpolation_t` nodes of a given type (and order) are already present.

4.2 Reading and Writing Interpolation Characteristics via the Family Interface

v2 text

The family contains the set of interpolation bases:

- for elements as a function of the element type `ElementType_t`;
- for the solution as a function of the combination `ElementType_t` and two interpolation orders `os` and `ot`. This means that no more than one specification can be present for the triplet $(t, \texttt{os}, \texttt{ot})$. The element type always refers back to the baseline element, i.e. the solution interpolation basis for the triplets (`TETRA_4`,4,0) and (`TETRA_10`,4,0) are the same;
- the number of coordinates of the control points is defined by the dimension associated to the element type.

Function signature	Modes
`cg_nelement_interpolation_read(fn, bn, fam, *ne)`	r--
`cg_element_interpolation_read(fn, bn, fam, en, name, *et)`	r--
`cg_element_interpolation_type_read(fn, bn, fam, en, *it)`	r--
`cg_element_interpolation_points_read(fn, bn, fam, en, *pu, *pv, *pw)`	r--
`cg_element_interpolation_write(fn, bn, fam, name, et, *en)`	-wm
`cg_element_interpolation_points_write(fn, bn, fam, en, *pu, *pv, *pw)`	-wm
`cg_element_isoparametric_write(fn, bn, fam, name, et, *en)`	-wm
`cg_nsolution_interpolation_read(fn, bn, fam, *ns)`	r--
`cg_solution_interpolation_read(fn, bn, fam, sn, name, *et, *os, *ot, *it)`	r--
`cg_solution_interpolation_points_read(fn, bn, fam, sn, *pu, *pv, *pw, *pt)`	r--
`cg_solution_interpolation_write(fn, bn, fam, name, et, os, ot, it, *sn)`	-wm
`cg_solution_interpolation_points_write(fn, bn, fam, sn, *pu, *pv, *pw, *pt)`	-wm
`cg_solution_interpolation_find(fn, bn, fam, et, os, ot, *sn, *it)`	r--
`cg_solution_interpolation_coefficients_read(fn, bn, fam, sn, *coeff)`	r--
`cg_solution_interpolation_coefficients_write(fn, bn, fam, sn, *coeff)`	-wm
`cg_element_interpolation_distribution_read(fn, bn, fam, en, *dist)`	r--
`cg_element_interpolation_distribution_write(fn, bn, fam, en, dist)`	-wm
`cg_solution_interpolation_distribution_read(fn, bn, fam, sn, *dist)`	r--
`cg_solution_interpolation_distribution_write(fn, bn, fam, sn, dist)`	-wm

Table 4: Family-level interpolation API functions.

Param	Description	I/O
`fn`	CGNS file index number	in
`bn`	Base index number	in
`fam`	Family index number	in
`ne`	Number of element interpolation blocks	in/out
`en`	Element interpolation index number	in
`ns`	Number of solution interpolation blocks	in/out
`sn`	Solution interpolation index number; returned as an output by `cg_solution_interpolation_find`	in/out
`os`	Spatial interpolation degree	in
`ot`	Temporal interpolation degree	in
`it`	Interpolation type (`InterpolationType_t`)	in/out
`pu`	Control points — u-coordinate	out
`pv`	Control points — v-coordinate (`NULL` for 1D)	out
`pw`	Control points — w-coordinate (`NULL` for 1D/2D)	out
`pt`	Control points — time coordinate (`NULL` if $q = 0$)	out
`coeff`	Monomial coefficients array (size from `cg_*_monomial_size`)	in/out
`dist`	Lagrange control point distribution (`LagrangeControlPointDistribution_t`)	in/out

Table 5: Input/output parameters for family-level API.

IsoParametric mesh interpolation. `cg_element_isoparametric_write` is a convenience writer that creates an `ElementInterpolation_t` node without a `LagrangeControlPoints` child. The interpolation type is implicitly `IsoParametric`: the element's own connectivity table coordinates are reused, and no control-point write call follows. The corresponding read function `cg_element_interpolation_points_read` returns `CG_NODE_NOT_FOUND` on such nodes.

Why the read call does not synthesise the points. `CG_NODE_NOT_FOUND` here is deliberate, not an omission. The call reports what the file contains, and that distinction carries information the caller needs: a writer that stores `LagrangeControlPoints` may place them anywhere unisolvent — Gauss-Lobatto-Legendre, for instance (Section 3.1.2) — whereas `IsoParametric` means the equidistant lattice of the standard node layout. If the read call returned synthesised convention points, a reader could no longer tell a file that specified its own distribution from one that did not, which is precisely the distinction the control-point array exists to record.

The locations themselves are never in doubt: they are the equidistant lattice on the reference domain of the element type (Section 6.7), traversed in the standard node order of Figure 1 (Section 3.2.1). Both are normative, so a reader can reconstruct the locations from the element type alone, with no further information from the file and no ambiguity.

That is not the same as saying the reader is well served. Reconstructing the lattice means tabulating the standard node ordering for every element family and order, which every reader implementing the full interpolation machinery would then do independently — duplicated work that a library is better placed to do once. A convenience helper returning that lattice for a given `ElementType_t`, alongside the existing element-type queries such as `cg_element_lagrange_interpolation_size`, is therefore a genuine candidate for inclusion rather than a hypothetical one. It is deliberately left open here: the choice is whether to specify it in this amendment or defer it, not whether it would be useful.

Modal coefficient access. `cg_solution_interpolation_coefficients_{read,write}` operate on the `MonomialCoefficients` child of a `SolutionInterpolation_t` node. The expected array length is given by `cg_solution_monomial_size`. These functions must not be called on a node whose interpolation type is `ParametricLagrange` or `IsoParametric`. There is no element-side counterpart: mesh interpolation is nodal only (Section 3.4.2), so no `ElementInterpolation_t` node carries modal coefficients.

4.3 Accessing Data in the `FlowSolution_t` Node

> **v2 text**
>
> The interpolation order is assigned per solution block within an unstructured zone, whereas the details concerning the interpolation functions are encoded in the family attached to the zone. In this case, the solution is supposed to be attached to `CellCenter`; the values are the expansion weights in the basis. The specific interpolation basis is defined through the combination of the element type and the interpolation orders.
>
> The cardinality of the interpolation functions is to be determined first by accessing the description of the interpolation.

Implementation note — superseded by Implementation note 2 (Section 3.4.5): the `CellCenter`

Function signature	Modes
`cg_sol_ptset_info(fn, bn, zn, sn, *ptype, *npnts)`	r--
`cg_sol_ptset_read(fn, bn, zn, sn, *pnts)`	r--
`cg_sol_ptset_write(fn, bn, zn, name, loc, ptype, npnts, *pnts, *sn)`	-wm
`cg_sol_interpolation_degree_read(fn, bn, zn, sn, *spatialDegree, *temporalDegree)`	r--
`cg_sol_interpolation_degree_write(fn, bn, zn, sn, spatialDegree, temporalDegree)`	-wm
`cg_sol_characteristic_length_read(fn, bn, zn, sn, *nscale, *numElements, *h_e)`	r--
`cg_sol_characteristic_length_write(fn, bn, zn, sn, nscale, numElements, *h_e)`	-wm

Table 6: FlowSolution-level interpolation degree and characteristic-length API.

Param	**Description**	**I/O**
`fn`	CGNS file index number	in
`bn`	Base index number	in
`zn`	Zone index number	in
`sn`	Solution block index number	in
`ptype`	Point set type (`PointRange` or `PointList`)	out
`npnts`	Number of points in the point set	out
`pnts`	Point set data	in/out
`loc`	Grid location (`InterpolationPoints` for high-order; both uniform and variable-order)	in
`spatialDegree`	Spatial interpolation degree	in/out
`temporalDegree`	Temporal interpolation degree	in/out
`nscale`	Number of normalisation factors per element: `1` selects the isotropic encoding, `PhysDim` the per-axis encoding	in/out
`numElements`	Number of elements covered by the solution block	in/out
`h_e`	Normalisation factors, total length `nscale`×`numElements` (`NULL` on read for a shape-only query)	in/out

Table 7: Input/output parameters for FlowSolution-level API.

location in the v2 text above is not used for high-order solutions in v3. All such `FlowSolution_t` nodes use `GridLocation = InterpolationPoints`.

4.4 Fortran Bindings

All CPEX-0045 functions are exposed to Fortran via Fortran 2003 `BIND(C)` interfaces in `cgns_f.F90`, with the conventional `_f` suffix (e.g. `cg_solution_interpolation_write_f`). Argument types follow the standard CGNS Fortran conventions: `integer(cgsize_t)` for index sizes, `integer(c_int)` for indices and enumerations, and `real(c_double)` for control point and coefficient arrays. A representative Fortran usage example is provided in `src/tests/test_high_orderf.F90`.

5 Extension to the SIDS File Mapping

5.1 `ElementInterpolation_t`: Child Node of `Family_t`

Family_t	
Children	
...	
name = <user defined> label = **`ElementInterpolation_t`** datatype = `I4` dimensions = 1 dimension values = 1 data = <element type> cardinality = 0:N	
Children	**Comments**
name = LagrangeControlPoints type = DataArray_t datatype = `R8` IndexDimension = 2 DataSize[] = [Dimension, NumberOfPoints] data = <point locations> cardinality = 0:1	`IndexDimension` is the rank of the array (2). `Dimension` is the spatial dimension of the element.
name = LagrangeControlPointDistribution datatype = **`LagrangeControlPointDistribution_t`** data = <choice of distribution> cardinality = 0:1	`GaussLobattoLegendre`, `Equidistant`, `GaussLegendre` or `WarpAndBlend`. Recommended, not required, when interpolation type is `ParametricLagrange`.

Table 8: Node structure for `ElementInterpolation_t`.

5.2 `SolutionInterpolation_t`: Child Node of `Family_t`

Family_t	
Children	
...	
name = <user defined> type = **`SolutionInterpolation_t`** datatype = `I4` dimensions = 1 dimension values = 3 data = <element type, spatial degree, temporal degree> cardinality = 0:N	
*The element type is the **`basic`** (linear) tag of the element family, normalised on write: `TETRA_10` and `TETRA_35` are both stored as `TETRA_4` (Section 3.4.4).*	
Children	**Comments**
name = InterpolationType datatype = **`InterpolationType_t`** data = <choice for interpolation type>	`ParametricLagrange`, `ParametricMonomialsPascal`, `CartesianMonomialsPascal` or `IsoParametric`
name = LagrangeControlPoints type = DataArray_t datatype = `R8` IndexDimension = 2 DataSize[] = [Dimension′, NumberOfPoints] data = <point locations> cardinality = 0:1	`IndexDimension` is the rank of the array (2). `Dimension`′ = element `Dimension` for purely spatial, or `Dimension`+1 for space-time (extra row holds parametric time τ). `NumberOfPoints` = $N_{\text{spatial}} \times (q+1)$ for space-time.
name = MonomialCoefficients datatype = `R8` data = <monomial coefficients> cardinality = 0:1	size from `cg_solution_monomial_size`. Present for `ParametricMonomialsPascal` or `CartesianMonomialsPascal`.
name = LagrangeControlPointDistribution datatype = **`LagrangeControlPointDistribution_t`** data = <choice of distribution> cardinality = 0:1	`GaussLobattoLegendre`, `Equidistant`, `GaussLegendre` or `WarpAndBlend`. Recommended, not required, when interpolation type is `ParametricLagrange`.

Table 9: Node structure for `SolutionInterpolation_t`.

5.3 `FlowSolution_t`: Added Child Node

A single child node is added to `FlowSolution_t`:

FlowSolution_t
Children
...
name = InterpolationDegrees type = **`IndexArray_t`** datatype = `I4` dimensions = 1 dimension values = 2 data = <spatial and temporal interpolation degree> cardinality = 0:1
name = CharacteristicLength type = **`DataArray_t`** datatype = `R8` dimensions = 1 *or* 2 (rank selects the normalisation) dimension values = $[\lvert\mathcal{E}\rvert]$ isotropic: one factor per element $[\texttt{PhysDim}, \lvert\mathcal{E}\rvert]$ per-axis: `PhysDim` factors per element, `PhysDim` fast-varying data = <per-element normalisation factors> cardinality = 0:1
Mandatory when the associated `SolutionInterpolation_t` has `InterpolationType = CartesianMonomialsPascal`; element ordering matches the field-array ordering. All factors must be strictly positive. $\lvert\mathcal{E}\rvert$ is the number of elements covered by the block. Writers record the factors they actually used; readers use the recorded values and do not recompute them (Section 3.3).

Table 10: New `InterpolationDegrees` and `CharacteristicLength` children of `FlowSolution_t`.

6 Implementation Specification

This section consolidates the formal constraints, on-disk encodings, error semantics, and default values that a conforming implementation must observe. It applies to both readers and writers.

6.1 Polynomial Degree and Geometric Order Limits

- **Geometric order** (encoded via `ElementType_t`): bounded by the available high-order tags, currently up to degree four (e.g. `HEXA_125`, the degree-4 tensor-product hexahedron, and `TETRA_35`, the degree-4 tetrahedron with $\binom{4+3}{3} = 35$ nodes), per Section 3.1. This is the unrenamed, pre-existing element-type concept, not `SpatialDegree`.
- **Solution spatial degree**: non-negative integer. The library writer (`cg_sol_interpolation_degree_write`) accepts any value ≥ 0. `cgnscheck` warns when the spatial degree falls outside the typical range $[0, 100]$, and in strict CPEX-0045 mode (`cgnscheck -s`) reports values outside $[0, 100]$ as errors.

- **Solution temporal degree**: non-negative integer. `cgnscheck` warns when outside $[0, 10]$; strict mode reports values outside $[0, 10]$ as errors.
- **No constraint couples the two.** Any combination of a non-negative `SpatialDegree` and a non-negative `TemporalDegree` is valid.

Degree zero is valid. `SpatialDegree` $= 0$ denotes a single spatial degree of freedom per element — a solution constant over the element — and must be accepted. It is the "standard interpolation (i.e. constant per element)" case of the v2 text (Section 3.4.4), and it is the natural representation of a finite-volume cell average. For a modal basis it is the single constant monomial, $N_{\text{modal}} = \binom{0+d}{d} = 1$. For a Lagrange basis it is the one control point recorded in `LagrangeControlPoints`, whose nodal function is identically one; the point set is unisolvent for P_0, so the basis is well defined (Section 3.1.2).

Consequently `TemporalDegree` > 0 with `SpatialDegree` $= 0$ is also valid, and describes a per-element value that is constant in space and varies in time — an unsteady finite-volume solution. Its degree-of-freedom count is $N_{\text{DOFs}} = 1 \times (q + 1) = q + 1$ per element, by the general rule of Section 6.2. Neither the writer nor `cgnscheck` may reject these configurations.

6.2 On-Disk Layout of Control Points and Coefficients

LagrangeControlPoints (purely spatial). The data array is stored as a 2D `R8` array with shape $[Dim, NPoints]$ in CGNS (Fortran-style) ordering, i.e. *Dim* is the fast-varying axis and *NPoints* the slow-varying axis. Equivalently, the byte stream is the concatenation of `NPoints` tuples, each of length *Dim*:

$$(u_0, v_0, w_0,\ u_1, v_1, w_1,\ \ldots,\ u_{N-1}, v_{N-1}, w_{N-1})$$

LagrangeControlPoints (space-time). For `SolutionInterpolation_t` with `TemporalDegree` > 0, the array shape becomes $[Dim + 1, NPoints_{\text{ST}}]$ where $NPoints_{\text{ST}} = NPoints_{\text{spatial}} \times (q + 1)$ and $q =$ `TemporalDegree`. Within each coordinate row, the points are ordered *time-major*:

$$\texttt{idx} = t \times N_{\text{spatial}} + s, \qquad t = 0, \ldots, q, \qquad s = 0, \ldots, N_{\text{spatial}} - 1$$

That is, all spatial points at time level $t = 0$ are listed first, then all spatial points at $t = 1$, and so on. Because the array has shape $[Dim + 1, NPoints_{\text{ST}}]$ in CGNS Fortran-style (column-major) ordering, the $Dim + 1$ dimension is the fast-varying axis. Consequently the byte stream is the concatenation of $NPoints_{\text{ST}}$ interleaved tuples, each of length $Dim + 1$:

$$(u_0, v_0, w_0, \tau_0,\ u_1, v_1, w_1, \tau_1,\ \ldots)$$

The spatial components and the temporal component τ are therefore *interleaved per point*; τ does *not* appear as a contiguous trailing block. Solution-data arrays in `FlowSolution_t` for space-time interpolation follow the same time-major ordering within each element's $N_{\text{DOFs}}(e)$ block.

MonomialCoefficients. A 1D `R8` array of length $N_{\text{modal}} = \binom{p+d}{d}$ for purely spatial interpolation, or $N_{\text{modal}} \times (q + 1)$ for space-time, where d is the element dimension, p is the spatial degree, and q is the temporal degree. The Pascal-traversal order is given by the loops in Section 3.2.3 and is a writer convention; the library does not validate ordering.

InterpolationDegrees. A 1D `IndexArray_t` (data type `I4`) of length 2, child of `FlowSolution_t`, holding [`spatialDegree`, `temporalDegree`]. Recognized only for `Unstructured` zones.

High-order `FlowSolution_t` field arrays. When `GridLocation = InterpolationPoints`, each field `DataArray_t` child of `FlowSolution_t` has length

$$L = \sum_{e \in \mathcal{E}} N_{\mathrm{DOFs}}(e),$$

where $\mathcal{E}$ is the set of elements covered by the block (all zone elements when no `PointRange` /`PointList` is present, or the listed elements otherwise) and $N_{\mathrm{DOFs}}(e)$ is the per-element DOF count from the matching `SolutionInterpolation_t` for element e. For a homogeneous block this collapses to $|\mathcal{E}| \cdot N_{\mathrm{DOFs}}$. Within the array, the DOFs of the first listed element are stored contiguously, followed by all DOFs of the second listed element, and so on:

$$\text{array index} = \Big(\sum_{k<i} N_{\mathrm{DOFs}}(e_k)\Big) + \mathit{local\_dof\_index}$$

where e_i is the i-th listed element (0-based) and *local_dof_index* runs from 0 to $N_{\mathrm{DOFs}}(e_i) - 1$ in the ordering established by the associated `SolutionInterpolation_t` (Lagrange control points listed in `LagrangeControlPoints`, or modal coefficients in the Pascal traversal of Section 3.2.3).

Metadata arrays are not solution fields. `CharacteristicLength` is a `DataArray_t` child of `FlowSolution_t`, but it is interpolation *metadata* and carries its own shape rather than the field length L above. Readers that enumerate the `DataArray_t` children of a `FlowSolution_t` to build the list of solution fields *must exclude it by name*, and must not apply the field size rule to it. A reader that does not exclude it will both report a spurious extra field and reject a conformant file, because the metadata length (`nscale` $\times\ |\mathcal{E}|$) does not equal L. Implementations should note that this failure is invisible on test files that contain no element sections, since field size checking cannot be performed at all in that case.

CharacteristicLength. An `R8` `DataArray_t` child of `FlowSolution_t` holding the per-element coordinate normalisation factors used by Cartesian modal interpolation (Section 3.3). The array rank selects the normalisation: rank 1 with shape [$|\mathcal{E}|$] is isotropic (one factor per element); rank 2 with shape [`PhysDim`, $|\mathcal{E}|$] is per-axis (`PhysDim` factors per element, `PhysDim` fast-varying, so one element's factors are contiguous). All factors must be strictly positive. Mandatory when the associated `SolutionInterpolation_t` uses `CartesianMonomialsPascal`; absent otherwise. Element ordering matches the field-array element ordering above. The values recorded are those the writer actually applied; readers must not recompute them from element geometry.

6.3 Validation Requirements (Reader)

A conforming reader must reject:

- `ElementInterpolation_t` payloads that are not `I4` scalar (the `ElementType_t` value);
- `SolutionInterpolation_t` payloads that are not `I4`, dimension 1, with `dim_vals[0] = 3` (the triple [*ElementType*, *SpatialDegree*, *TemporalDegree*]);

- `LagrangeControlPoints` children whose data type is not `R8` or whose array dimension is not 2;
- `MonomialCoefficients` children whose data type is not `R8` or whose array dimension is not 1;
- `DataArray_t` children of `SolutionInterpolation_t` whose name is anything other than `LagrangeControlPoints` or `MonomialCoefficients`;
- `DataArray_t` children of `ElementInterpolation_t` whose name is anything other than `LagrangeControlPoints` — in particular a `MonomialCoefficients` child, which mesh interpolation never carries (Section 3.4.2);
- `SolutionInterpolation_t` nodes that do not contain exactly one `InterpolationType_t` child named `InterpolationType`;
- more than one `LagrangeControlPointDistribution_t` child under a single parent, or one whose data is not a value named in Section 3.1.2;
- Duplicate `ElementInterpolation_t` nodes (same `ElementType_t` tag) or duplicate `SolutionInterpolation_t` nodes (same basic-type, `os`, `ot`) within a family;
- `CharacteristicLength` children whose data type is not `R8`, whose array rank is neither 1 nor 2, whose `nscale` (rank-2 only) is neither 1 nor `PhysDim`, whose element count disagrees with the block, or which contain a non-positive factor.

A conforming reader must also *exclude* `CharacteristicLength` from the solution-field list of its parent `FlowSolution_t`, and must not apply the field size rule of Section 6.2 to it.

A mismatch between `LagrangeControlPointDistribution` and the coordinates in `LagrangeControlPoints` is *not* grounds for rejection. The coordinates are authoritative (Section 3.1.2): a conforming reader must build the basis from them and should warn about the disagreement, rather than failing the file or adopting the named family's nodes.

6.4 Cross-Validation Rules (Writer)

- `cg_solution_interpolation_points_write` must be rejected when the parent `SolutionInterpolation_t` has `InterpolationType` of `ParametricMonomialsPascal` or `CartesianMonomialsPascal` (modal nodes do not carry Lagrange points; coefficients must be written instead).
- `cg_element_isoparametric_write` must not be followed by `cg_element_interpolation_points_write` on the same node.
- Re-writing an existing `LagrangeControlPoints` or `MonomialCoefficients` array requires the file to be opened in `CG_MODE_MODIFY`; in `CG_MODE_WRITE` the second write must be rejected.
- A writer must not record a `LagrangeControlPointDistribution` whose family nodes disagree with the `LagrangeControlPoints` coordinates written on the same node. `cgnscheck` compares the two whenever both are present and reports a mismatch — an error under `cgnscheck -s`, a warning otherwise. Because the coordinates are authoritative (Section 3.1.2), the remedy is to correct or omit the name, never to alter the coordinates to match it.

- `cg_sol_interpolation_degree_write` requires `GridLocation = InterpolationPoints` on the parent `FlowSolution_t`. For backward compatibility with files written under earlier drafts, the library *also* accepts `GridLocation = CellCenter` provided a `PointRange` or `PointList` is present, but `cgnscheck -s` flags this combination as non-conformant and conformant writers must use `InterpolationPoints` exclusively.

6.5 Error-Code Semantics

Code	Meaning
`CG_OK`	Operation succeeded.
`CG_NODE_NOT_FOUND`	Returned by `cg_element_interpolation_points_read` and `cg_solution_interpolation_points_read` when the corresponding node has no `LagrangeControlPoints` child (i.e. the interpolation type is `IsoParametric` or modal); also returned by `cg_sol_interpolation_degree_read` when the `FlowSolution_t` has no `InterpolationDegrees` child. Callers must treat this as a normal control-flow signal, not an error.
`CG_ERROR`	Validation failure or I/O error. Use `cg_get_error()` to retrieve the diagnostic message string.

Table 11: Error-code semantics for CPEX-0045 functions.

Detecting interpolation type at runtime. For `ElementInterpolation_t`, call `cg_element_interpolation_type_read`; the returned value is either `ParametricLagrange` or `IsoParametric`. Neither modal type is ever returned for mesh interpolation (Section 3.4.2). For `SolutionInterpolation_t`, the type is the `InterpolationType` child read by `cg_solution_interpolation_read`.

6.6 Default Values

- `SolutionInterpolation_t`: `SpatialDegree` is required (no default); `TemporalDegree` defaults to 0.
- `FlowSolution_t.InterpolationDegrees`: when the child is absent `cg_sol_interpolation_degree_read` returns `CG_NODE_NOT_FOUND` with output values `(0, 0)`. Such a `FlowSolution_t` is treated as a standard (non-high-order) solution.

6.7 Coordinate-System Conventions

All supported elements use the *bi-unit* reference domain, i.e. each parametric coordinate ranges over $[-1, 1]$ rather than $[0, 1]$. These are the standard reference elements of the high-order literature [5], and they are the domains drawn in Figure 1, which is normative:

- Tensor-product elements (`BAR`, `QUAD`, `HEXA`): the bi-unit cube $[-1, 1]^d$.
- Triangle (`TRI`): $\{(u, v) : u \geq -1,\ v \geq -1,\ u + v \leq 0\}$, with vertices $(-1, -1)$, $(1, -1)$, $(-1, 1)$. The hypotenuse is the line $u + v = 0$.
- Tetrahedron (`TETRA`): $\{(u, v, w) : u, v, w \geq -1,\ u + v + w \leq -1\}$, with vertices $(-1, -1, -1)$, $(1, -1, -1)$, $(-1, 1, -1)$, $(-1, -1, 1)$. The oblique face is the plane $u + v + w = -1$.

- Prism (`PENTA`): the `TRI` cross-section above, in (u, v), extruded along $w \in [-1, 1]$.
- Pyramid (`PYRA`): $\{(u, v, w) : -1 \le w \le 1,\ |u| \le \frac{1-w}{2},\ |v| \le \frac{1-w}{2}\}$ — the square base $[-1, 1]^2$ at $w = -1$ contracting linearly to the apex $(0, 0, 1)$. The functional space on this domain is not a polynomial space and is defined in [3]; its cardinality is given in Section 3.2.2.

This is an interoperability requirement, not a recommendation. Control points are stored as bare parametric coordinates, so a writer and a reader that assume different reference domains will exchange a file that is structurally valid and passes `cgnscheck` while being geometrically wrong. A $[0, 1]$-based simplex convention in particular is *not* conformant: its coordinates are indistinguishable from valid bi-unit coordinates, so the error cannot be detected after the fact. The library does *not* enforce these ranges on user-supplied control points — any `double` value is accepted — which is precisely why conformance is the writer's responsibility.

6.8 Scope of `InterpolationPoints` Grid Location

`GridLocation_t = InterpolationPoints` is valid only on `FlowSolution_t` nodes. `ArbitraryGridMotion_t` and `DiscreteData_t` nodes must not use this location.

7 Variable-Order Configurations and IsoParametric Solutions

This section describes the three usage patterns that arise when different cells or different field variables require different polynomial orders, and the use of `IsoParametric` for solution interpolation.

7.1 Variable Order Per Cell (Disjoint `PointRange`)

When different subsets of cells use different polynomial orders, create one `FlowSolution_t` block per subset using `cg_sol_ptset_write` with non-overlapping `PointRange` intervals, each annotated with `cg_sol_interpolation_degree_write`:

```
/* Cells 1..4 at order 2, cells 5..8 at order 3 */
cgsize_t range_p2[2] = {1, 4};
cg_sol_ptset_write(fn,B,Z,"FS_p2",InterpolationPoints,PointRange,2,
   range_p2,&S1);
cg_sol_interpolation_degree_write(fn,B,Z,S1, 2,0);

cgsize_t range_p3[2] = {5, 8};
cg_sol_ptset_write(fn,B,Z,"FS_p3",InterpolationPoints,PointRange,2,
   range_p3,&S2);
cg_sol_interpolation_degree_write(fn,B,Z,S2, 3,0);
```

7.2 Variable Order Per Cell (`PointList`)

Non-contiguous subsets require `PointList` instead of `PointRange`:

```
cgsize_t odd[4]  = {1,3,5,7};
cgsize_t even[4] = {2,4,6,8};
cg_sol_ptset_write(fn,B,Z,"FS_Odd", InterpolationPoints,PointList,4,odd
   , &S1);
```

```
cg_sol_interpolation_degree_write(fn,B,Z,S1, 2,0);
cg_sol_ptset_write(fn,B,Z,"FS_Even",InterpolationPoints,PointList,4,
   even,&S2);
cg_sol_interpolation_degree_write(fn,B,Z,S2, 3,0);
```

7.3 Variable Order Per Field Variable

When different physical fields use different polynomial orders over the *same* set of cells, create one `FlowSolution_t` per field variable with the same `PointRange` but distinct interpolation orders:

```
cgsize_t all[2] = {1, N_ELEM};
/* Density at order 2 */
cg_sol_ptset_write(fn,B,Z,"FS_Density",InterpolationPoints,PointRange
   ,2,all,&S1);
cg_sol_interpolation_degree_write(fn,B,Z,S1, 2,0);
cg_field_write(fn,B,Z,S1,RealDouble,"Density",density,&fld);

/* VelocityX at order 3 */
cg_sol_ptset_write(fn,B,Z,"FS_VelocityX",InterpolationPoints,PointRange
   ,2,all,&S2);
cg_sol_interpolation_degree_write(fn,B,Z,S2, 3,0);
cg_field_write(fn,B,Z,S2,RealDouble,"VelocityX",velocity,&fld);
```

7.4 `IsoParametric` for `SolutionInterpolation_t`

When `InterpolationType` is set to `IsoParametric`, the solution uses the same interpolation basis as the mesh (i.e. the `ElementInterpolation_t` defined for the same element type in the family). No `LagrangeControlPoints` or `MonomialCoefficients` child is stored. The corresponding `ElementInterpolation_t` must exist in the family.

```
int sn;
cg_solution_interpolation_write(fn,bn,fam,"QUAD4_IsoParam",
    QUAD_4, /*os=*/1, /*ot=*/0, IsoParametric, &sn);
/* No cg_solution_interpolation_points_write call needed */
```

Reading points from an `IsoParametric` solution. Unlike the mesh case (Section 4.2), the control points of an `IsoParametric` *solution* are present in the file: the referenced `ElementInterpolation_t` must exist, and it either stores `LagrangeControlPoints` or is itself `IsoParametric`. `cg_solution_interpolation_points_read` therefore *resolves the reference* and returns the control points of that node, rather than reporting the solution node's own lack of a data array. It returns `CG_NODE_NOT_FOUND` only when the referenced `ElementInterpolation_t` is itself `IsoParametric`, in which case no control points are stored anywhere and the reference domain of Section 6.7 with the node order of Figure 1 applies.

Implementation status: this resolution is specified here but not yet implemented on the `CPEX45_-high_order_wip` branch, where the call still reports `CG_NODE_NOT_FOUND` from the solution node's

own missing array. Like the coordinate-versus-distribution comparison of Section 3.1.2, this is a normative statement the reference implementation has yet to catch up with.

Returning the referenced points is not synthesis: the data is in the file, one documented indirection away, and the indirection is exactly what `IsoParametric` means. A caller that needs to know the interpolation type asks for it directly through `cg_solution_interpolation_read`, so nothing is lost by having the points call succeed.

8 Compliance Tests

A conforming implementation must pass the following objective tests. These are independent of any particular library and can be applied directly to a written file.

Partition of unity (Lagrange). For `ParametricLagrange` interpolation of any supported element type and order p,

$$\sum_{i=1}^{N} \lambda_i(\mathbf{u}) = 1 \quad \text{for any } \mathbf{u} \text{ in the parametric domain,} \tag{9}$$

within tolerance $\epsilon = 10^{-12}$. Reference test points, each strictly interior to its reference domain (Section 6.7): `QUAD` $p = 2$ at $(u, v) = (0.3, -0.7)$; `HEXA` $p = 2$ at $(0.5, -0.5, 0.2)$; `TRI` $p = 2$ at $(-0.5, -0.3)$, for which $u + v = -0.8 < 0$; `TETRA` $p = 2$ at $(-0.6, -0.5, -0.4)$, for which $u + v + w = -1.5 < -1$.

Kronecker delta (Lagrange). At each control point $\mathbf{u}_j$,

$$\lambda_i(\mathbf{u}_j) = \delta_{ij} \tag{10}$$

within tolerance 10^{-12} for all $i, j \in \{1, \ldots, N\}$.

Modal coefficient ordering. Represent a known polynomial in the chosen modal basis and verify that the resulting coefficient vector matches the expected ordering of Section 3.2.3 (parametric Pascal traversal) or Section 3.3 (Cartesian column-major). For example, the polynomial $f(u, v) = u^2 + v$ over a `QUAD` $p = 2$ Cartesian modal basis must produce a coefficient vector whose only non-zero entries are $c[u^2v^0] = 1$ and $c[u^0v^1] = 1$; if the indices of those non-zero entries differ from the convention, the implementation is non-compliant.

Geometry reconstruction. For a curved `QUAD_9` element with the centre node displaced from its parametric midpoint, evaluating the interpolated geometry at $(u, v) = (0, 0)$ must reproduce the displaced centre coordinates within tolerance 10^{-10}.

Numerical stability (Cartesian modal). For a `QUAD` element whose physical extents span large or small scales (e.g. 10^6 or 10^{-6} in either coordinate), modal-coefficient writes followed by reads must round-trip the data without overflow or precision loss when the writer applies the normalization of Section 3.3. Failure of this test indicates missing or incorrect normalization at the writer. The reader must additionally verify that the normalisation factors recovered from the `CharacteristicLength` node correctly reconstruct the unnormalised coordinate space without precision loss, for both the isotropic (rank-1) and per-axis (rank-2) encodings.

Anisotropic conditioning (Cartesian modal, per-axis). For a strongly stretched element — e.g. a `QUAD` or `HEXA` of aspect ratio 10^4, representative of a wall-resolved boundary-layer cell — a per-axis `CharacteristicLength` must round-trip exactly and must place every normalised coordinate in $O([-1, 1])$. An implementation that emits a single isotropic factor for such an element is conformant only if it accepts the resulting loss of relative precision in the short direction; per-axis normalisation is the recommended encoding in this case.

Compliance checklist.

- Partition-of-unity test for every supported element type and order.
- Kronecker delta property at every control point.
- Modal-coefficient ordering matches Pascal traversal (parametric) or column-major tensor product (Cartesian).
- Geometry reconstruction within tolerance 10^{-10}.
- Numerical stability for extreme coordinate scales.
- Time-major ordering for space-time data layout.
- Principal vertices identified per the linear sub-element rule (Cartesian modal only).
- Coordinate normalization applied for Cartesian modal interpolation, with the factors actually used recorded in `CharacteristicLength`.
- Both `CharacteristicLength` encodings handled on read: rank-1 isotropic and rank-2 per-axis.
- Lagrange basis constructed from the stored `LagrangeControlPoints` coordinates whether or not a `LagrangeControlPointDistribution` attribute is present, and no distribution assumed when it is absent.

References

[1] CPEX 0038: "Quartic Elements for High Order".

[2] CPEX 0036: "Cubic Elements for High Order".

[3] M. Bergot, G. Cohen, and M. Duruflé, "Higher-order Finite Elements for Hybrid Meshes Using New Nodal Pyramidal Elements", *Journal of Scientific Computing*, vol. 42, pp. 345–381, 2010. doi:10.1007/s10915-009-9334-9

[4] T. Warburton, "An explicit construction of interpolation nodes on the simplex", *Journal of Engineering Mathematics*, vol. 56, no. 3, pp. 247–262, 2006. doi:10.1007/s10665-006-9086-6

[5] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Springer Texts in Applied Mathematics, vol. 54, Springer, 2008. doi:10.1007/978-0-387-72067-8

[6] H. Luo, J. D. Baum, and R. Löhner, "A discontinuous Galerkin method based on a Taylor basis for the compressible flows on arbitrary grids", *Journal of Computational Physics*, vol. 227, no. 20, pp. 8875–8893, 2008. doi:10.1016/j.jcp.2008.06.035

[7] D. Kuzmin, "Entropy stabilization and property-preserving limiters for discontinuous Galerkin discretizations of nonlinear hyperbolic equations", *Journal of Numerical Mathematics*, vol. 29, no. 4, pp. 307–322, 2021. arXiv:2004.03521

[8] S. Massago, *mathex* library, part of the Small Scientific Library (SSCILIB). http://sscilib.sourceforge.net