CAE Plugin SDK

A Basic How-to

CAE Plugin Runtime Header Files

To define your plugin's capabilities, you must edit the configuration header files that were copied to your plugin’s project folder.

At runtime, this information is loaded by Pointwise. It is used by the GUI and is accessible through glyph2 scripts*.

- site.h
- rtPwpVersions.h
- rtPwpPluginInfo.h
- rtPwpInitItems.h
- rtCaepSupportData.h
- rtCaepInitItems.h
- rtCaepInstanceData.h

* See the pw::Application/CAE Solver Attributes documentation in the glyph2 manual for more information.

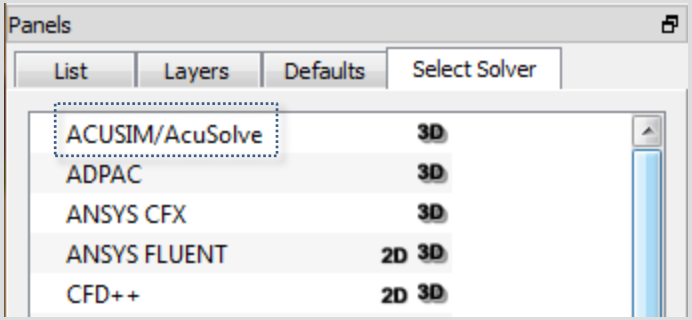
site.h

PWP_SITE_GROUPID

- Globally unique, 16-bit, integer site-identifier value.
- Combined with the plugin's id value to construct a globally unique id (GUID).
- If you plan on releasing a plugin outside your site, you need to obtain a **PWP_SITE_GROUPID** value from Pointwise support.
- Pointwise will not load plugins properly if GUID conflicts are detected.

PWP_SITE_GROUPNAME

- A string representing your company or group.
- All Pointwise developed plugins use the group name "Pointwise".
- Pointwise uses this name to distinguish between similarly named plugins from different authors (group ids).
- A CAE plugin's full name is ***GroupName/CaeName*** (e.g. Alpha/CGNS, Alpha/xml, Beta/CGNS, Beta/xml).
- Defaults to **PWP_GROUPNAME_DEFAULT**.



rtPwpVersions.h

VERSION_PWP_MAJOR

VERSION_PWP_MINOR

- The PWP-API major/minor version values.
- Pointwise uses these values to determine plugin compatibility.
- Typically, you will not change these values. They are set correctly for a given SDK release.

VERSION_LIB_MAJOR

VERSION_LIB_MINOR

- The software release major/minor version value.
- Typically, you WILL change these values every time a new plugin version is released.
- The version scheme is determined by the plugin author.
- Pointwise does not use these values directly.

rtPwpPluginInfo.h

- Currently, not used by the Pointwise GUI.
- You should only edit the company, support, and copyright strings.
- May be any suitable text.

```
VERSION_PWP_INIT,          // conforms to this PWP-API version
VERSION_LIB_INIT,          // software library release version
"Pointwise, Inc.",         // company/author description
"support.pointwise.com",   // support description (phone, web-link).
"Copyright(c) 2008-2011", // copyright description
0,                         // number of APIs (set at runtime)
0,                         // default msg callback (set at runtime)
0,                         // spy msg callback (set at runtime)
```

rtPwpInitItems.h

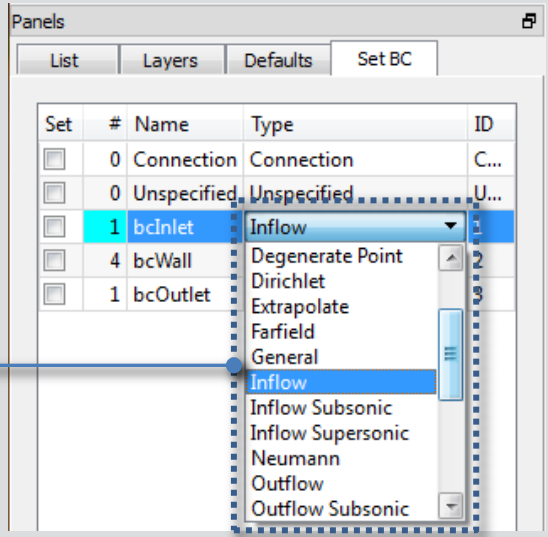
- Pointwise uses these values to determine the plugin type(s).
- Typically, you will not change these values. They are set correctly for a given SDK release.

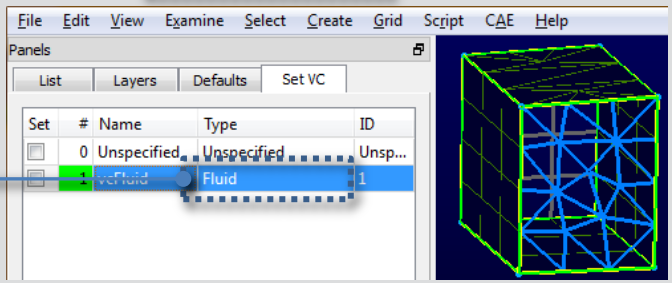
```
//.............................
{
    {PWP_API_PLUGIN "/1.0", VERSION_PWP_INIT},
    0,
},
//.............................
{
    {CAEP_API_EXPORT "/1.0", {1,0}},
},
```

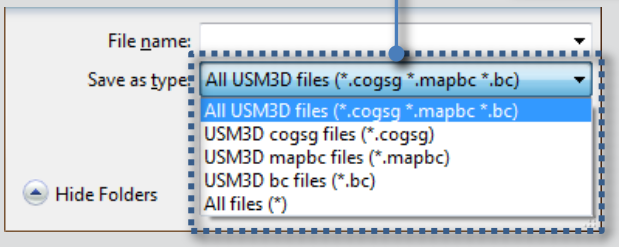
rtCaepSupportData.h

- This file is used to define the valid BCs, VCs, and file extensions.
- There will be one entry for each BC, VC, and file extension.

```
CAEP_BCINFO caepMyPluginBCInfo[] = {
    { “inflow”,  100 },
    { “outflow”, 101 },
    { “wall”,    102 },
};
/*-------------------------------------*/
CAEP_VCINFO caepMyPluginVCInfo[] = {
    { “Fluid”,  200 },
    { “Solid”,  201 },
    { “Plasma”, 202 },
};
/*-------------------------------------*/
const char * caepMyPluginFileExt[] = {
    “cogsg”,
    “mapbc”,
    “bc”
};
```



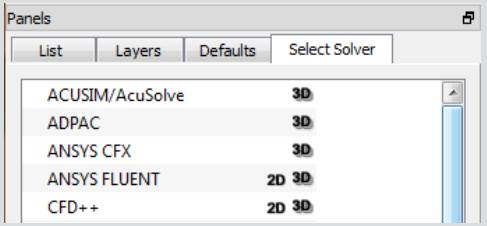


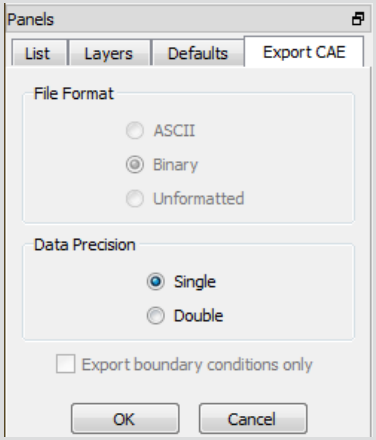


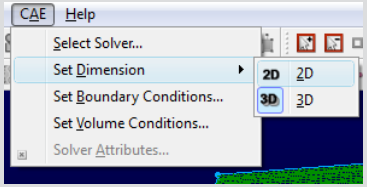
rtCaepInitItems.h

- Pointwise uses these values to configure its functionality.
- Typically, you will make many changes to this file.
- This data is passed to **runtimeWrite(CAEP_RTITEM *pRti, ...)**.

```
#define ID_CaeMyPlugin     2  // unique to your site
//== CAEP_RTITEM *pRti ================================
{
    //== CAEP_FORMATINFO FormatInfo ====================================
    {   PWP_SITE_GROUPNAME,          // const char *group (see site.h)
        "Your Solver Name“,          // const char *name
        MAKEGUID(ID_CaeMyPlugin),    // PWP_UINT32 id

        PWP_FILEDEST_FOLDER,         // CAEP_ENUM_FILEDEST fileDest

        PWP_FALSE,                   // PWP_BOOL allowedExportConditionsOnly
        PWP_TRUE,                    // PWP_BOOL allowedVolumeConditions

        PWP_TRUE,                    // PWP_BOOL allowedFileFormatASCII
        PWP_FALSE,                   // PWP_BOOL allowedFileFormatBinary
        PWP_FALSE,                   // PWP_BOOL allowedFileFormatUnformatted

        PWP_FALSE,                   // PWP_BOOL allowedDataPrecisionSingle
        PWP_TRUE,                    // PWP_BOOL allowedDataPrecisionDouble

        PWP_FALSE,                   // PWP_BOOL allowedDimension2D
        PWP_TRUE                     // PWP_BOOL allowedDimension3D
    },
    // ---SNIP---
}
```







rtCaepInitItems.h
(continued)

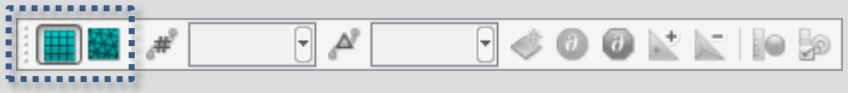
- You will NOT make any changes to this section.
- These values are defined in **rtCaepSupportData.h**.

```
#define ID_CaeMyPlugin     2
//== CAEP_RTITEM *pRti ================================
{
    // ---SNIP---

    &pwpRtItem[1],  // PWU_RTITEM*

    //== CAEP_BCINFO*    pBCInfo;  -- array of format BCs or NULL
    //== PWP_UINT32      BCCnt;    -- # format BCs
    caepMyPluginBCInfo,                 // CAEP_BCINFO*
    ARRAYSIZE(caepMyPluginBCInfo),      // PWP_UINT32 BCCnt

    //== CAEP_VCINFO*    pVCInfo;  -- array of format VCs or NULL
    //== PWP_UINT32      VCCnt;    -- # format VCs
    caepMyPluginVCInfo,                 // CAEP_VCINFO* pVCInfo
    ARRAYSIZE(caepMyPluginVCInfo),      // PWP_UINT32 VCCnt

    //== const char**    pFileExt; -- array of valid file extensions
    //== PWP_UINT32      ExtCnt;   -- # valid file extensions
    caepMyPluginFileExt,                // const char **pFileExt
    ARRAYSIZE(caepMyPluginFileExt),     // PWP_UINT32 ExtCnt

    // ---SNIP---
}
```

rtCaepInitItems.h
(continued)

- This section specifies the supported solver element types.
- Set to **PWP_TRUE** if an element type is supported.
- Set to **PWP_FALSE** if an element type is *not* supported.

```
#define ID_CaeMyPlugin     2
//== CAEP_RTITEM *pRti ===================================
{
    // ---SNIP---

    //== PWP_BOOL  elemType[PWGM_ELEMTYPE_SIZE]; ===========
    {   PWP_TRUE,           // elemType[PWGM_ELEMTYPE_BAR]
        PWP_TRUE,           // elemType[PWGM_ELEMTYPE_HEX]
        PWP_TRUE,           // elemType[PWGM_ELEMTYPE_QUAD]
        PWP_TRUE,           // elemType[PWGM_ELEMTYPE_TRI]
        PWP_TRUE,           // elemType[PWGM_ELEMTYPE_TET]
        PWP_TRUE,           // elemType[PWGM_ELEMTYPE_WEDGE]
        PWP_TRUE },         // elemType[PWGM_ELEMTYPE_PYRAMID]

    // ---SNIP---
}
```



rtCaepInitItems.h
(continued)

- You will NOT make any changes to this section.
- These values are set and managed by the SDK at runtime.

```
#define ID_CaeMyPlugin     2
//== CAEP_RTITEM *pRti ================================
{
    // ---SNIP---

    0,              // FILE *fp

    //== PWU_UNFDATA UnfData ========================
    {   0,          // PWP_UINT32 status
        0,          // FILE *fp
        0,          // sysFILEPOS fPos
        PWP_FALSE,  // PWP_BOOL hadError
        PWP_FALSE,  // PWP_BOOL inRec
        0,          // PWP_UINT32 recBytes
        0,          // PWP_UINT32 totRecBytes
        0    },     // PWP_UINT32 recCnt

    0,              // PWGM_HGRIDMODEL model
    0,              // const CAEP_WRITEINFO *pWriteInfo
    0,              // PWP_UINT32 progTotal
    0,              // PWP_UINT32 progComplete
    {0},            // clock_t clocks[]
    0,              // PWP_BOOL opAborted

    // ---SNIP---
}
```

rtCaepInitItems.h
(continued)

- This section initializes programmer-defined runtime instance data.
- Only needed if data members were added in `rtCaepInstanceData.h`.

```
#define ID_CaeMyPlugin     2
//== CAEP_RTITEM *pRti ===================================
{
    // ---SNIP---

    // If you added any custom data in rtCaepInstanceData.h, you need to
    // initialize it here. The commented out block below matches the
    // example MY_CAEP_DATA struct given in rtCaepInstanceData.h
    {   0,
        0,
        0.0,
        "string" },
}
```

rtCaepInstanceData.h

- This file is used to Customize the CAEP_RTITEM declaration.
- Plugins often need to track temporary, global data during export.
- Your data is passed to **`runtimeWrite(CAEP_RTITEM *pRti, ...)`** as part of **`pRti`**.
- You can pass **`pRti`** to other functions without having to clutter the function call prototype.

There are 2 approaches to adding custom data members to CAEP_RTITEM.

1. Add a single **`struct`** containing all your custom data members.
2. Add each custom data member as a separate value.

rtCaepInstanceData.h (continued)

Single Struct Instance Data

In rtCaepInstanceData.h add:

```
typedef struct MY_CAEP_DATA_t {
    int data1;
    int data2;
    float data3;
    char *pStr;
}
MY_CAEP_DATA;

#define CAEP_RUNTIME_INSTDATADECL   MY_CAEP_DATA myData;
```

At runtime, you access your data as:

```
PWP_BOOL runtimeWrite(CAEP_RTITEM *pRti, ...snip...)
{
    pRti->myData.data1;
    pRti->myData.data2;
    pRti->myData.data3;
    pRti->myData.pStr;
}
```

rtCaepInstanceData.h (continued)

Separate Values Instance Data

In rtCaepInstanceData.h add:

```
#define CAEP_RUNTIME_INSTDATADECL  int    data1; \
                                   int    data2; \
                                   float  data3; \
                                   char  *pStr;
```

At runtime, you access your data as:

```
PWP_BOOL runtimeWrite(CAEP_RTITEM *pRti, ...snip...)
{
    pRti->data1;
    pRti->data2;
    pRti->data3;
    pRti->pStr;
}
```

Implement the runtimeWrite() Function

- A simple function body is in the file **runtimeWrite.cxx**
  - located in the **.../PluginSDK/src/plugins/CaeMyPlugin/** folder*.
- When a CAE export is initiated, Pointwise will:
  1. Initialize the plugin.
  2. Populate a **CAEP_RTITEM** structure.
  3. Prepare a **PWGM_HGRIDMODEL** database.
  4. Populate a **CAEP_WRITEINFO** structure.
  5. Call **runtimeWrite(**`CAEP_RTITEM *pRti, PWGM_HGRIDMODEL model, const CAEP_WRITEINFO *pWriteInfo`**).**
- Access runtime data using the **pRti** param.
- Access grid data using the **model** param.
- Access export options using the **pWriteInfo** param.

* for a plugin project created using: `% mkplugin -uns CaeMyPlugin`

Useful CAEP_RTITEM data

CAEP_BCINFO *pBCInfo;
PWP_UINT32 BCCnt;
The BC data from rtCaepSupportData.h

CAEP_VCINFO *pVCInfo;
PWP_UINT32 VCCnt;
The VC data from rtCaepSupportData.h

const char **pFileExt;
PWP_UINT32 ExtCnt;
The file extensions from rtCaepSupportData.h

PWP_BOOL elemType[PWGM_ELEMTYPE_SIZE];
The supported element types from rtCaepSupportData.h

Useful CAEP_RTITEM data (continued)

`FILE *fp;`

A file pointer. For plugins specifying `PWP_FILEDEST_FILENAME`, a file is automatically opened before `runtimeWrite()` is called and closed after `runtimeWrite()` returns. fp contains the pointer to this opened file. For all other `PWP_FILEDEST` types, fp will be `NULL`. The CAE exporter is allowed to manipulate `fp` using the `pwpFileXxxx()` functions as needed (see pwpPlatform.h).

`PWU_UNFDATA unfData;`

Unformatted FORTRAN file I/O data buffer. For exporters requiring this file format, this data buffer can be used with the `PwuUnfXxxx()` helper functions (see apiPWPUtils.h).

`PWGM_HGRIDMODEL model;`

Handle to the grid model being exported. This is the same handle passed into `runtimeWrite()`.

`const CAEP_WRITEINFO *pWriteInfo;`

The `CAEP_WRITEINFO` data for this export. This is the same pointer passed into `runtimeWrite()`.

Useful CAEP_WRITEINFO data

`const char * fileDest`

Requested file destination. This will be either a full file name, a base file name, or a folder name (depending on the PWP_FILEDEST_XXX type specified in rtCaepInitItems.h).

`PWP_BOOL conditionsOnly`

Set to PWP_TRUE if only conditions are to be exported (no grid data).

`CAEP_ENUM_ENCODING encoding`

Requested export encoding. One of `PWP_ENCODING_ASCII`, `PWP_ENCODING_BINARY`, `PWP_ENCODING_UNFORMATTED`.

`CAEP_ENUM_PRECISION precision`

Requested export precision. One of `PWP_PRECISION_SINGLE`, `PWP_PRECISION_DOUBLE`.

`CAEP_ENUM_DIMENSION dimension`

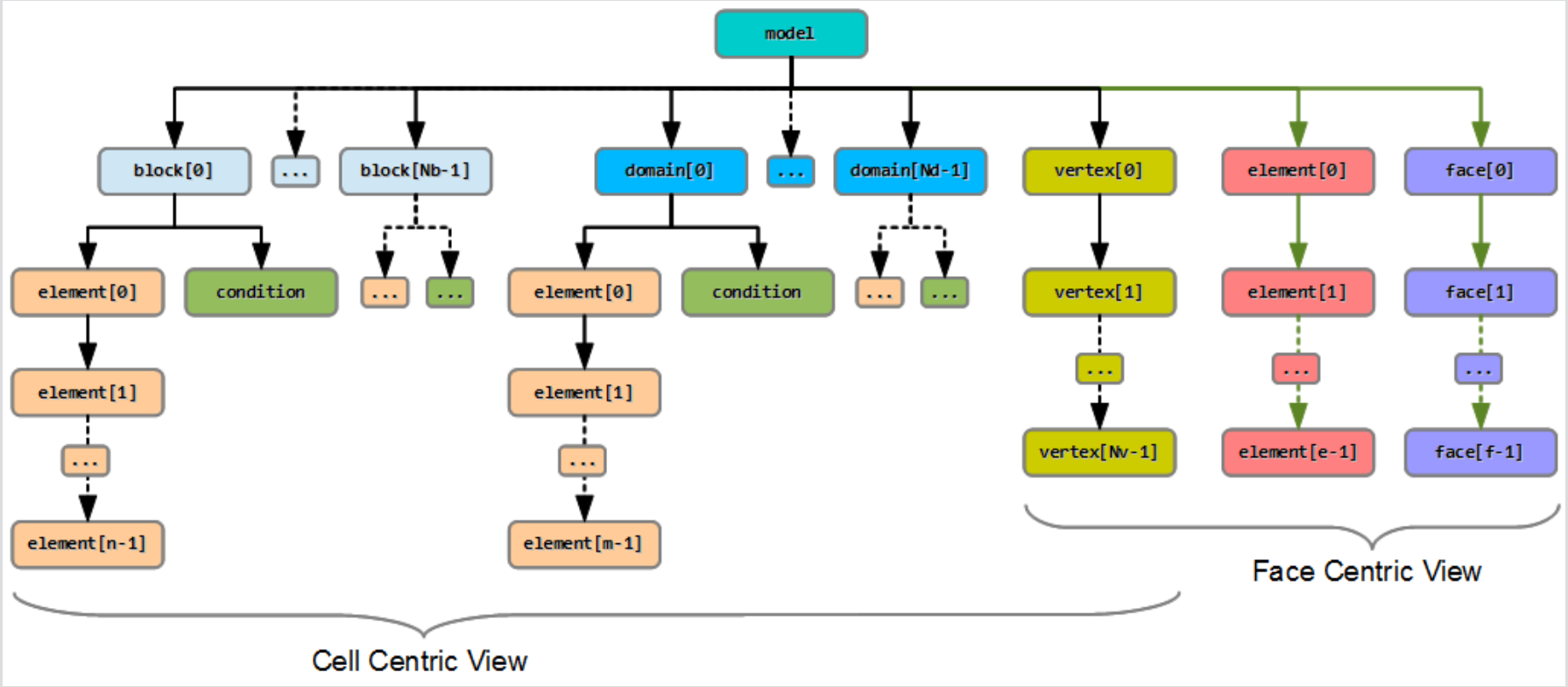
The dimensionality of the grid being exported. Plugins that support both 2D and 3D grids will need to check this value. One of `PWP_DIMENSION_2D`, `PWP_DIMENSION_3D`.

The Grid Model

- What is a handle?
  - It is an *opaque*, *un-typed* data value.
  - It is an alias for the actual data object.
  - Used to populate a local copy of an object's data.
- Why use handles instead of pointers?
  - Minimize the opportunity for accidental corruption of Pointwise data.
  - Minimize the opportunity for intentional corruption of Pointwise data.
  - Eliminate memory allocations and management.
  - Provides a generic interface for both 2D and 3D grids.
- How do I use a handle?
  - Declare a local data buffer.
  - Obtain a handle.
  - Get the data associated with the handle.

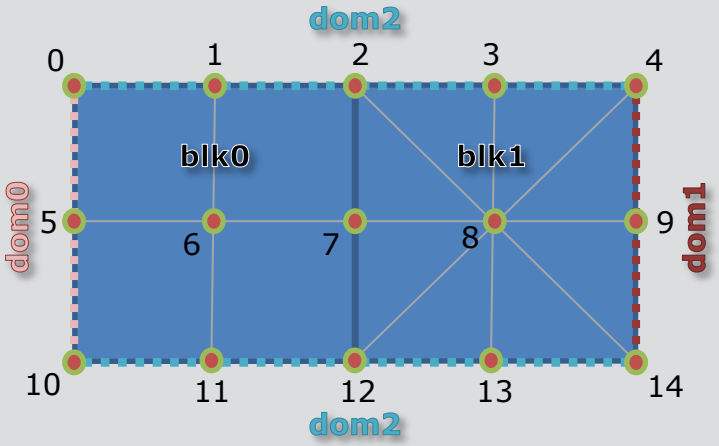
The Grid Model
(continued)

Handle Hierarchy for Unstructured Grids



Example Unstructured Grid Model

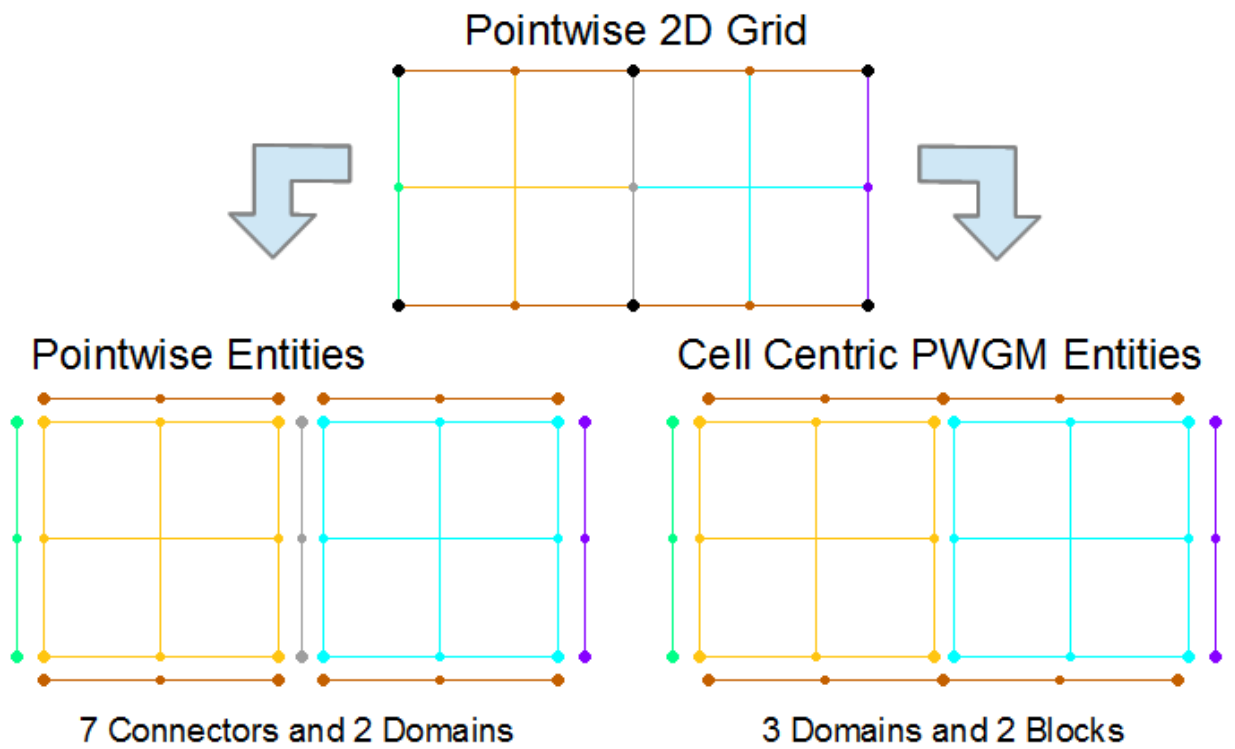
Cell Centric Access




```
PWGM_HGRIDMODEL model;
PwModVertexCount(model); // returns 15
PwModBlockCount(model);  // returns 2
PwModDomainCount(model); // returns 3

PWGM_ELEMCOUNTS counts;
PWGM_HBLOCK blk0, blk1;
PwBlkElementCount(blk0, &counts); // returns 4
PWGM_ECNT_Quad(counts);     // == 4

PwBlkElementCount(blk1, &counts); // returns 8
PWGM_ECNT_Tri(counts);      // == 8

PWGM_HDOMAIN dom0, dom1, dom2;
PwDomElementCount(dom0, &counts); // returns 2
PWGM_ECNT_Bar(counts);      // == 2

PwDomElementCount(dom1, &counts); // returns 2
PWGM_ECNT_Bar(counts);      // == 2

PwDomElementCount(dom2, &counts); // returns 8
PWGM_ECNT_Bar(counts);      // == 8
```

Example Unstructured Grid Model

Cell Centric Access (continued)



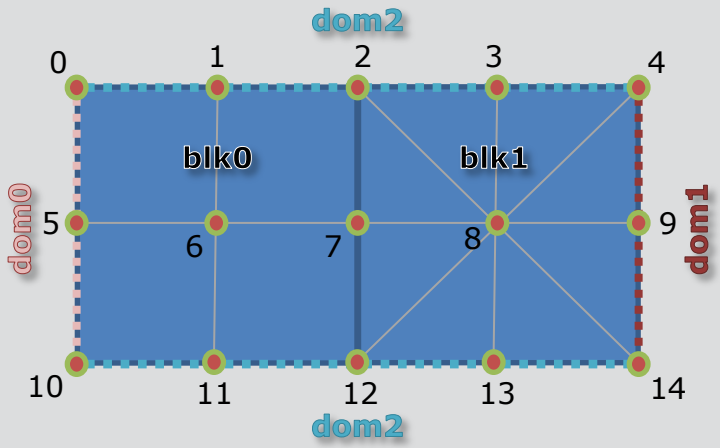
Entities of the same color share the same condition.

Even though the upper and lower brown entities are not physically connected, they are merged into a single boundary in the PWGM.

The gray entity between the blocks is a connection and is not part of any PWGM boundary.

Example Unstructured Grid Model

Face Centric Access

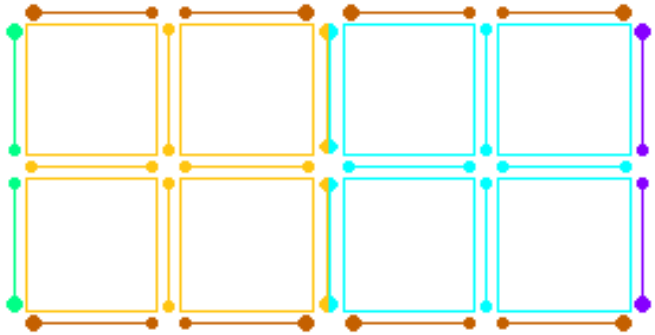



```
PWGM_HGRIDMODEL model;
PWGM_ELEMCOUNTS counts;
struct MyData myData = { /* init */ };

PwModVertexCount(model); // returns 15

// Set block element enumeration order to all Tri's
// first followed by all Quad's
PwModAppendEnumElementOrder(model, PWGM_ELEMTYPE_TRI);
PwModAppendEnumElementOrder(model, PWGM_ELEMTYPE_QUAD);

PWP_UINT32 ndx;
PWP_UINT32 cnt = PwModEnumElementCount(model, &counts);
PWGM_ECNT_Tri(counts);    // == 8 (ndx = 0 to 7)
PWGM_ECNT_Quad(counts);   // == 4 (ndx = 8 to 11)
For (ndx = 0; ndx < cnt; ++ndx) { // cnt == 12
    PwModEnumElements(model, ndx);
}

// Stream cell faces to the beginCB(), faceCB() and
// endCB() callbacks
PwModStreamFaces(model, PWGM_FACEORDER_BOUNDARYFIRST,
   beginCB, faceCB, endCB, &myData);
```

Example Unstructured Grid Model

Face Centric Access (continued)

Face Centric PWGM Entities



8 Cell Elements and 22 Face Elements

Face elements that only touch one cell element are boundary face elements.

Boundary face elements of the same color share the same boundary condition.

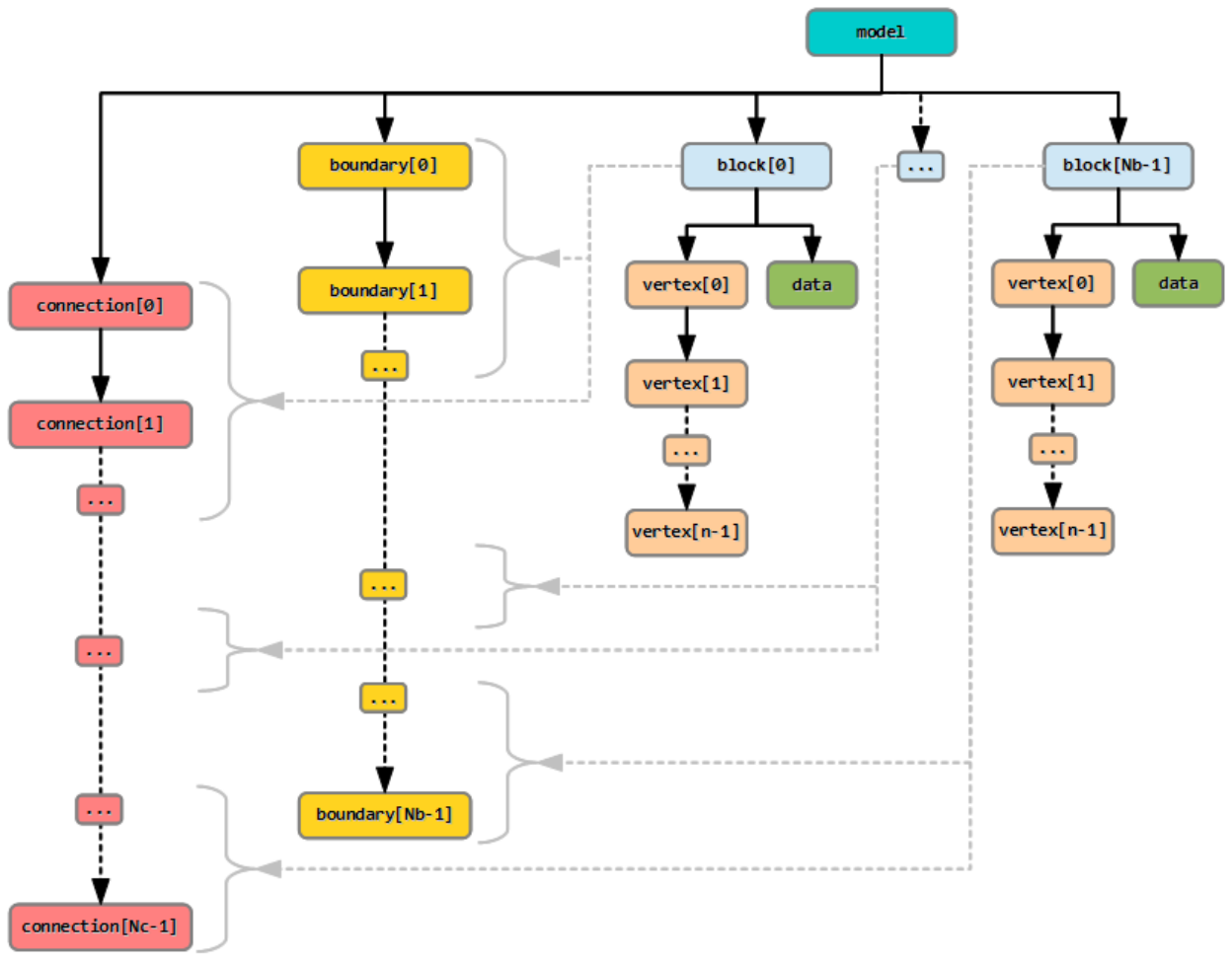
Cell elements of the same color share the same volume condition.

The single-colored face elements between two cell elements of the same color are internal block faces.

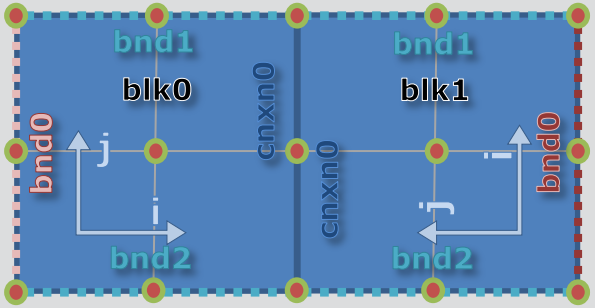
The bi-colored face elements between two cell elements with different colors are internal, block connection faces.

The Grid Model
(continued)

Handle Hierarchy for Structured Grids



Example Structured Grid Model




```
PWGM_HGRIDMODEL model;
PwModBlockCount(model);  // returns 2
PwBlkSize(blk0, &size);
   // size(i,j) = (3,3)
PWGM_INDEX3 ndx3 = { 0, 1 }; // i, j
PWGM_VERTDATA vData;
PwBlkNdxVertData(blk0, ndx3, &vData);
   // vData.x == 0.0 , .y == 0.5, .z == 0.0
```

```
PWGM_BNDRYDATA bData;
PWGM_CONDDATA bcData;
PwModBoundaryCount(model); // returns 6
PwModNdxBoundaryAndCondition(model, 3,
        &bData , &bcData);
   // bData.block = blk1, .face1 = PWGM_FACE_JMIN
   // bcData.name = "outlet", .id = 22
   // bcData.type = "Outflow", .tid = 201
```

```
PWGM_CNXNDATA cData;
PwModConnectionCount(model); // returns 2
PwModNdxConnection(model, 0, &cData);
   // cData.block1 = blk0, .face1 = PWGM_FACE_IMAX
   // cData.block2 = blk1, .face2 = PWGM_FACE_JMAX
PwBlkConnectionCount(blk1); // returns 1
PwBlkNdxConnection(blk1, 0, &cData);
   // cData.block1 = blk1, .face1 = PWGM_FACE_JMAX
   // cData.block2 = blk0, .face2 = PWGM_FACE_IMAX
```

The Grid Model
(continued)

Grid Model Functions (docs)

Common Functions
- PwModBlockCount
- PwModEnumBlocks

Unstructured Functions
- PwModDomainCount
- PwModEnumDomains
- PwModEnumVertices
- PwModVertexCount
- PwBlkElementCount
- PwBlkEnumElements
- PwBlkCondition
- PwDomElementCount
- PwDomEnumElements
- PwDomCondition
- PwVertDataMod
- PwVertIndexMod
- PwVertXyzVal
- PwElemDataMod
- PwModStreamFaces
- PwModEnumElementCount
- PwModEnumElements
- PwModAppendEnumElementOrder
- PwModDefaultEnumElementOrder

Structured Functions
- PwBlkSize
- PwBlock
- PwBlkNdxVertData
- PwModConnectionCount
- PwModEnumConnections
- PwConnection
- PwModNdxConnection
- PwModBoundaryCount
- PwModEnumBoundaries
- PwBoundary
- PwBndryCondition
- PwModNdxBoundary
- PwModNdxBoundaryAndCondition
- PwBlkBoundaryCount
- PwBlkEnumBoundaries
- PwBlkNdxBoundary
- PwBlkNdxBoundaryAndCondition
- PwBlkConnectionCount
- PwBlkEnumConnections
- PwBlkNdxConnection

Structured Utility Functions
- PwXform2to3
- PwXform3to2
- PwXformApply
- PwXformFollows
- PwXform2Apply
- PwXform2Follows
- PwInRange

Example Code

```
// 2 equivalent approaches to accessing a set of grid handles

// using an “unknown count” and a while loop
PWP_UINT32 ndx = 0;
PWGM_HBLOCK block = PwModEnumBlocks(model, ndx);
while (PWGM_HBLOCK_ISVALID(block)) {
    if (!writeBlock(pRti, block, condOnly)) {
        break;
    }
    block = PwModEnumBlocks(model, ++ndx);
}

// using an “explicit count” and a for loop
PWP_UINT32 cnt = PwModBlockCount(model);
for (ndx = 0; ndx < cnt; ++ndx) {
    block = PwModEnumBlocks(model, ++ndx);
    if (!writeBlock(pRti, block, condOnly)) {
        break;
    }
}
```

Example Code

```
// write unstructured block using cell centric view
static int
writeUnstructuredBlock(CAEP_RTITEM *pRti, PWGM_HBLOCK block, PWP_BOOL condOnly)
{
    int ret = PWGM_HBLOCK_ISVALID(block);
    if (ret) {
        PWGM_ELEMDATA elemData;
        PWGM_CONDDATA condData;
        PWGM_ELEMCOUNTS elemCnts;
        PWP_UINT32 eCnt = PwBlkElementCount(block, &elemCnts);
        if (PwBlkCondition(block, &condData)) {
            writeCondData(pRti, &condData);
        }
        if (!condOnly) {
            writeElemCounts(pRti, &elemCnts);
            eCnt = 0;
            while (PwElemDataMod(PwBlkEnumElements(block, eCnt++), &elemData)) {
                writeElemData(pRti, &elemData);
            }
        }
    }
    return ret;
}
```

Example Code

```
#include “MyData.h”

// write unstructured grid using face centric view
static void
writeUnstructuredGrid(CAEP_RTITEM *pRti)
{
    // set cell ordering and then stream faces
    if (setCellOrder(pRti)) {
        MyData myData;
        PWGM_ENUM_FACEORDER faceOrder = PWGM_FACEORDER_BOUNDARYFIRST;
        if (PwModStreamFaces(pRti->model, faceOrder, beginCB, faceCB, endCB, &myData)) {
            PWGM_ELEMCOUNTS counts;
            PWP_UINT32 cnt = PwModEnumElementCount(pRti->model, &counts);
            // enumerate global grid cells
            for (PWP_UINT32 ndx = 0; ndx < cnt; ++ndx) {
                PWGM_HELEMENT hElem = PwModEnumElements(pRti->model, ndx);
                // write element info (cells in 3D, tris/quads in 2D)
            }
        }
    }
}
```

Example Code

```
bool setCellOrder(CAEP_RTITEM *pRti)
{
    // all tets, all hexes, all wedges+pyramids in an unspecified sequence
    return PwModAppendEnumElementOrder(pRti->model, PWGM_ELEMTYPE_TET) &&
           PwModAppendEnumElementOrder(pRti->model, PWGM_ELEMTYPE_HEX);
}

PWP_UINT32 beginCB(PWGM_BEGINSTREAM_DATA *data)
{
    MyData *myData = (MyData*)data->userData;
    // prepare for face stream, write header info, etc.
}

PWP_UINT32 faceCB(PWGM_FACESTREAM_DATA *data)
{
    MyData *myData = (MyData*)data->userData;
    // write face info (tris/quads in 3D, edges in 2D)
}

PWP_UINT32 endCB(PWGM_ENDSTREAM_DATA *data)
{
    MyData *myData = (MyData*)data->userData;
    // cleanup after face stream, write footer info, etc.
}
```

Example Code

```
// write structured block
static PWP_BOOL
writeStructuredBlock(CAEP_RTITEM *pRti, PWGM_HBLOCK block)
{
    PWP_BOOL ret = PWP_FALSE;
    PWGM_STR_SIZE size;
    PWGM_BLOCKDATA blkData;
    if (!PWGM_HBLOCK_ISVALID(block)) {
    }
    else if (PwBlkSize(block, &size) && PwBlock(block, &blkData)) {
        if (0 == blkData.name) {
            blkData.name = "";
        }
        if (CAEPU_RT_DIM_3D(pRti)) {
            // do 3D logic
        }
        else {
            // do 2D logic
        }
        ret = !pRti->opAborted && writeBlkVertices(pRti, block);
        ret = !pRti->opAborted && writeBlkConnections(pRti, block);
        ret = !pRti->opAborted && writeBlkBoundaries(pRti, block);
    }
    return ret;
}
```