Sphere

Skeleton for PHysical and Engineering REsearch

Ver 2.0.0

コンフィグレーションクラスマニュアル

2010 年 8 月 01 日

目次

１．　概要

SPHERE フレームワークは XML 形式のコンフィグレーションファイルに記述された設定を読み込むことで起動し制御されます。

コンフィグレーションファイルに記述される XML 要素には 2 種類あり、SPHERE では「SPHERE 定義要素」と「ユーザ定義要素」が存在します。

「SPHERE 定義要素」は SPHERE があらかじめ用意している要素で、記述される情報（属性値）の意味は固定です。
これらの要素は SPHERE フレームワークによって使用され、ソルバー開発者が直接読み取りを行う必要はありません。
「SPHERE 定義要素」には以下の要素があります。

SphDomainInfo
SphSteer
SphFileList
その他については、コンフィグレーション文法マニュアル参照

「ユーザ定義要素」は、"SphUserDifine"要素の配下に記述します。
"SphUserDifine"要素の配下に記述する要素名、属性名、階層構造はソルバー開発者が任意に定めることができます。

本マニュアルでは、SPHERE に実装されたソルバーがコンフィグレーションファイルに記述された「ユーザ定義要素」から設定値を取得する方法について説明します。

また、V-Sphere 形式のコンフィグレーションファイルへのアクセスも可能となっています。V-Sphere 形式のコンフィグレーションからの設定値の取得方法については後半に記述しています。

２．　コンフィグレーションへのアクセス

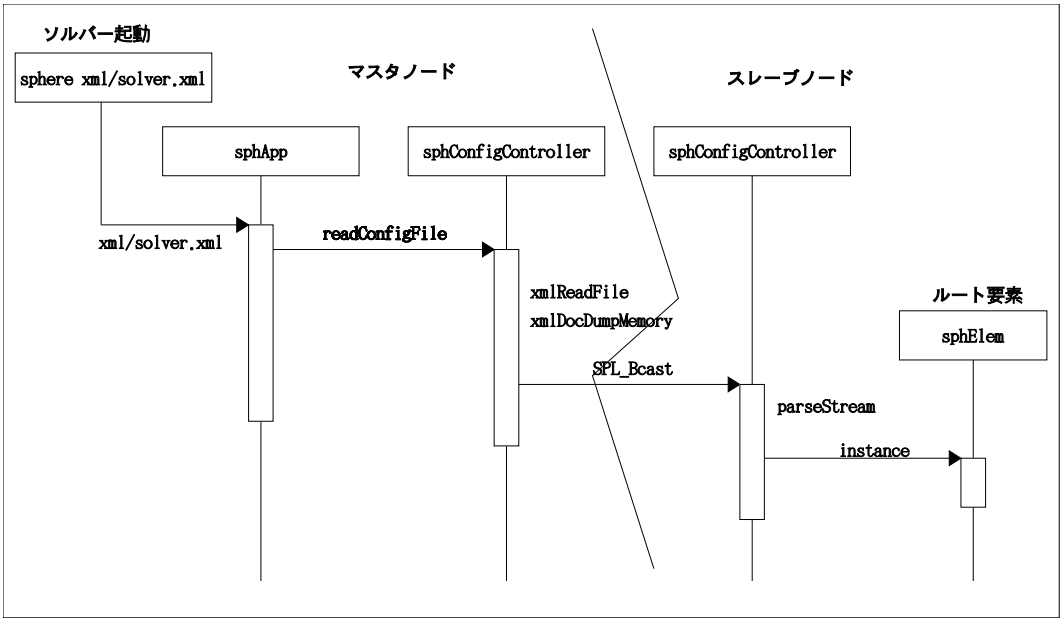
２．１　　コンフィグレーションファイル読込処理

コンフィグレーションファイルの指定は SPHERE の起動時に起動引数として指定します。

```
> sphere xml/solver.xml
```

sphere　　ソルバー実行モジュール

xml/solver.xml　コンフィグレーションファイル



(1) コンフィグレーションファイルは"sphConfigController::readConfigFile"メソッドにより読み込まれ、xmlDoc を生成します。
(2) 更に、コンフィグレーションに記述されている外部 XML ファイルを読み込み、１つの xmlDoc ツリーを作成します。
(3) 作成した xmlDoc は"xmlDocDumpMemory"関数により xmlDoc から xmlChar に変換した後に、xmlChar を全ノードに対してブロードキャストを行います。
(4) xmlDoc のストリームである xmlChar を受信したノードは受信ストリームから xmlDoc に変換します。
(5) すべてのノードは xmlDoc をパースして sphElem によるツリーを生成します。
(6) このとき生成される sphElem はコンフィグレーションのルート要素となります。

２．２　ソルバクラスからコンフィグレーション要素へのアクセス

SPHERE が生成するソルバークラス、フロークラス、カプラークラスには生成を行ったコンフィグレーション要素がすべてのクラスに設定されています。
ここでの"生成を行ったコンフィグレーション要素"とは読込を行ったコンフィグレーションファイルのルート要素ではなく、各クラスの制御が記述されている要素のことです。
ソルバークラスであれば SphSolver 要素、フロークラスは SphFlow 要素、カプラークラスは SphCoupler 要素となります。

（コンフィグレーションと生成クラス例）

```
<?xml version="1.0"?>
<SphereConfig version="2.0.0">

  <SphFlow   id="1" label="FLOW01" class="FLOW01" >        フロークラス要素
        ....
  </SphFlow>

  <SphSolverList>
    <SphSolver id="1" label="solv01" class="solv01">              ソルバー01 クラス要素
        ....
    </SphSolver>

    <SphSolver id="2" label="solv02" class="solv02">              ソルバー02 クラス要素
        ....
    </SphSolver>
  </SphSolverList>

  <SphCouplerList>
    <SphCoupler id="1" label="coupl01"   class="coupl01">         カプラークラス要素
        ....
    </SphCoupler>
  </SphCouplerList>

  <SphFileList> ....   </SphFileList>
</SphereConfig
```

各クラスのコンフィグレーション要素を取得するには以下のメソッドを使用します。

const sphElem* getConfigElem() const	
コンフィグレーション中の各自のソルバー、フロー、カプラーが担当（制御）する要素を取得します。	
引数	なし
戻り値	担当（制御）要素

（使用例）

```
int sphSolverSolv01::initializeSolver(sphStepTime* steptime)
{
```

```
    // ソルバー要素の取得
    const sphElem* solv_elem = this->getConfigElem();
    // SphUserDifine 要素の取得
    const sphElem* user_elem = getElem(SPH_CFG_NAME_USERDEFINE);
}
```

getConfigElem()によって取得した要素からユーザ定義要素("SphUserDifine")内に記述したパラメータの取得は sphElem クラスの要素・属性の取得メソッドを使用します。

２．３　　パラメータの取得

２．３．１　大文字・小文字の区別

sphElem クラスの要素・属性メソッドに指定する要素名、属性名、属性値については大文字・小文字を区別しません。

２．３．２　外部 XML ファイルへのアクセス

コンフィグレーションファイルに sph_xml_config_file 要素を記述することにより、コンフィグレーション要素のツリー構成に外部 XML ファイルの要素が追加することができます。

（コンフィグレーション例）

コンフィグレーションファイル

```
<SphSolver>
    <SphUserDefile>
        <table>
            <sph_xml_config_file file_name="./material.xml"/>          <Material_Table>追加
        </table>
    </SphUserDefile>
</SphSolver>
```

material.xml

```
<Material_Table>
    <Elem name="Fluid" id="1" comment="Water">
        <Param name="density" dtype="REAL" value="998.2" />
        <Param name="specific_heat" dtype="REAL" value="4182" />
        <Param name="thermal_conductivity" dtype="REAL" value="598e-3" />
    </Elem>
</Material_Table>
```

ツリー構成

```
<SphSolver>
    <SphUserDefile>
        <table>
            <Material_Table>                    sph_xml_config_file 要素が Material_Table と入れ替え
                <Elem name="Fluid" id="1" comment="Water">
                    <Param name="density" dtype="REAL" value="998.2" />
                    <Param name="specific_heat" dtype="REAL" value="4182" />
                    <Param name="thermal_conductivity" dtype="REAL" value="598e-3" />
                </Elem>
            </Material_Table>
        </table>
    </SphUserDefile>
</SphSolver>
```

<Param name="density">要素へのアクセスコード

```
// SphSolver
const sphElem* solv_elem = this->getConfigElem();
// SphUserDefile
const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
// table
const sphElem* tbl_elem = user_elem->getElem("table");
// Material_Table
const sphElem* mattbl_elem = tbl_elem->getElem("Material_Table");
// Elem
const sphElem* elem_elem = mattbl_elem->getChildElemFirst("Elem");
// Param
const sphElem* param_elem = elem_elem->getChildElemFirst("Param");
const char* name = param_elem->getAttrString("name");
float value = param_elem->getAttrReal4("value");
printf("name=%s, value=%f¥n",name, value);
```

sph_xml_config_file 要素によって記述された外部 XML ファイルへのアクセスは、sph_xml_config_file 要素部分に外部 XML ファイルのルート要素が挿入され sph_xml_config_file 要素は存在しないものとしてアクセスします。

３．　メソッド一覧

No	所属クラス	メソッド名	説明
1	sphSolverBase sphFlowBase sphCouplerbase	const sphElem* getConfigElem() const	各自のソルバー、フロー、カプラーが担当（制御）する要素を取得します。
2	sphElem	const sphElem* getElem(　　const char* elem_name) 　　const;	子要素名を指定して子要素を取得します。
3	sphElem	const sphElem* getElem(　　unsigned int num, 　　const char* elem_name00, 　　...) const;	子孫要素名を指定して子孫要素を取得します。
4	sphElem	const sphElem* getChildElemFirst(　　const char* elem_name = NULL) 　　const;	子要素名を指定して最初の子要素を順次取得します。
6	sphElem	const sphElem* getChildElemNext(　　const sphElem* elem, 　　const char* elem_name = NULL) 　　const;	子要素名を指定して次の子要素を順次取得します。
5	sphElem	const sphElem* getChildElemFirst(　　unsigned int num, 　　const char* elem_name00, 　　...)　const;	子孫要素名を指定して最初の子孫要素を順次取得します。
7	sphElem	const sphElem* getChildElemNext(　　const sphElem* elem, 　　unsigned int num, 　　const char* elem_name00, 　　...)　const;	子孫要素名を指定して次の子孫要素を順次取得します。

No	所属クラス	メソッド名	説明
8	sphElem	int countNumOfElem(const char* elem_name = NULL) const;	子要素名を指定して子要素数を取得します。
9	sphElem	int countNumOfElem(unsigned int num, const char* elem_name00, ...) const;	子孫要素名を指定して子孫要素数を取得します。
10	sphElem	const sphElem* getChildElemByAttr(const char* elem_name, const char* attr_name, const char* attr_value) const;	子要素名、属性名、属性値と一致する子要素を取得します。
11	sphElem	const char* getElemName() const;	要素名を取得します。
12	sphElem	const char* getAttrString(const char* name) const;	属性値を char*で取得します。
13	sphElem	short getAttrShort (const char* name) const;	属性値を short で取得します。
14	sphElem	int getAttrInt (const char* name) const;	属性値を int で取得します。
15	sphElem	long getAttrLong (const char* name) const;	属性値を long で取得します。
16	sphElem	long long getAttrLLong (const char* name) const;	属性値を long long で取得します。
17	sphElem	float getAttrReal4 (const char* name) const;	属性値を float で取得します。

No	所属クラス	メソッド名	説明
18	sphElem	double getAttrReal8 (const char* name) const;	属性値を double で取得します。
19	sphElem	SphDCType getAttrSphDcType (const char* name) const;	属性値を SphDCType 型で取得します。
20	sphElem	SPL_Datatype getAttrSplDataType (const char* name) const;	属性値を SPL_Datatype 型で取得します。
21	sphElem	SphCrdDef getAttrSphCrdDef(const char* name) const;	属性値を SphCrdDef 型で取得します。
22	sphElem	bool isAttrValue (const char* name, bool default_value = true) const;	属性値を bool で取得します。
23	sphElem	bool existAttribute(const char* name) const;	属性名が存在するかチェックします。

４．　メソッド詳細

４．１　自クラスのコンフィグレーションの取得

const sphElem* getConfigElem() const	
コンフィグレーション中の各自のソルバー、フロー、カプラーが担当（制御）する要素を取得します。	
引数	なし
戻り値	担当（制御）要素

生成されたソルバークラス、フロークラス、カプラークラスが担当するコンフィグレーション要素を取得します。

（使用例：子要素の取得参照）

４．２　子要素の取得

const sphElem* getElem(const char* elem_name) const;		
子要素名を指定して子要素を取得します。		
引数	elem_name	取得を行う子要素名
戻り値	子要素	

指定要素の子要素名を指定して最初に見つかった子要素を取得します。
指定要素配下に取得を行う子要素が１つだけに特定できる場合のみ使用します。

（使用例）

```
int sphSolverSolv01::initializeSolver(sphStepTime* steptime)
{
    // ソルバー要素の取得
    const sphElem* solv_elem = this->getConfigElem();
    // SphUserDifine 要素の取得
    const sphElem* user_elem = getElem(SPH_CFG_NAME_USERDEFINE);
}
```

４．３　子孫要素の取得

const sphElem* getElem(unsigned int num, const char* elem_name00, ...) const;		
子孫要素名を指定して子孫要素を取得します。		
引数	num	指定要素名の数
	elem_name00	最初の子要素名
	...	２番目以降の子要素名
戻り値	子孫要素	

指定要素の子要素名を複数指定して指定数分階層を下った要素を取得します。
指定要素配下に取得を行う子孫要素が１つだけに特定できる場合のみ使用します。

（使用例）

コンフィグレーション例

```
<SphSolver>
    <SphUserDefile>
        <Initial_State>
            <Density value="1.25e00" />
        </Initial_State>
    </SphUserDefile>
</SphSolver>
```

コード例

```
// ソルバー要素の取得
const sphElem* solv_elem = getConfigElem();
// SphUserDifine 要素の取得
const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
// Initial_State/Density 要素の取得
const sphElem* den_elem = user_elem->getElem(2, "Initial_State", "Density");
```

４．４　子要素の順次取得（最初の要素）

const sphElem*getChildElemFirst(const char* elem_name = NULL) const;	
子要素名を指定して最初の子要素を順次取得します。	
引数	elem_name　　子要素名
戻り値	子要素

指定要素の子要素名を指定して最初に見つかった子要素を取得します
指定要素配下に取得を行う子要素が複数存在する場合、getChildElemNext と組み合わせて要素を順次取得します。
子要素名を省略した場合は、要素名に関係なく順次子要素を先頭から取得します。

（使用例：getChildElemNext 参照）

４．５　子要素の順次取得（次の要素）

const sphElem* getChildElemNext(const sphElem* elem,const char* elem_name = NULL) const;
子要素名を指定して次の子要素を順次取得します。

引数	elem	前回取得要素
	elem_name	子要素名
戻り値	子要素	

指定要素の子要素名を指定して次に見つかった子要素を取得します

指定要素配下に取得を行う子要素が複数存在する場合、getChildElemFirst と組み合わせて要素を順次取得します。

getChildElemNext で指定する子要素名は getChildElemFirst と同じ子要素名でなければなりません。

子要素名を省略した場合は、要素名に関係なく順次子要素を先頭から取得します。

（使用例）

コンフィグレーション例

```
<SphSolver>
    <SphUserDefile>
        <Solid id="600" comment="Fe">
            <density   value="7870.0" />
        </Solid>
        <Solid id="500" comment="Al">
            <density   value="2688.0" />
        </Solid>
        <Solid id="255" comment="Fe">
            <density   value="7870.0" />
        </Solid>
    </SphUserDefile>
</SphSolver>
```

コード例

```
    // ソルバー要素の取得
    const sphElem* solv_elem = getConfigElem();
    // SphUserDifine 要素の取得
    const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
    // Solid[@id=600]要素の取得
    const sphElem* solid_elem = user_elem->getChildElemFirst("Solid");
    printf("id = %d, value = %f¥n",
                        solid_elem->getAttrInt("id"),
                        solid_elem->getElem("density")->getAttrReal4("value"));
    while (solid_elem != NULL) {
```

```
        // Solid[@id=500], Solid[@id=255]要素の取得
        solid_elem = user_elem->getChildElemFirst(solid_elem, "Solid");
        if (solid_elem != NULL) {
            printf("id = %d, value = %f¥n",
                        solid_elem->getAttrInt("id"),
                        solid_elem->getElem("density")->getAttrReal4("value"));
        }
    }
```

４．６　子孫要素の順次取得（最初の要素）

const sphElem* getChildElemFirst(unsigned int num, const char* elem_name00, ...)　const;		
子孫要素名を指定して最初の子孫要素を順次取得します。		
引数	num	指定要素名の数
	elem_name00	最初の子要素名
	...	２番目以降の子要素名
戻り値	子孫要素	

指定要素の子要素名を複数指定して指定数分階層を下った要素を取得します。
指定要素配下に取得を行う子孫要素が複数存在する場合、getChildElemNext と組み合わせて要素を順次取得します。

（使用例：getChildElemNext 参照）

４．７　子孫要素の順次取得（次の要素）

const sphElem* getChildElemNext(const sphElem* elem, unsigned int num, const char* elem_name00,...)　const;		
子孫要素名を指定して次の子孫要素を順次取得します。		
引数	elem	前回取得要素
	num	指定要素名の数
	elem_name00	最初の子要素名
	...	２番目以降の子要素名
戻り値	子孫要素	

指定要素の子孫要素名を指定して次に見つかった子孫要素を取得します
指定要素配下に取得を行う子孫要素が複数存在する場合、getChildElemFirst と組み合わせて要素を順次取得します。
getChildElemFirst で取得した子孫要素と同一階層、同一親要素の次の子孫要素を取得します。
getChildElemNext で指定する子孫要素名は getChildElemFirst と同じ子孫要素名でなければなりません。

（使用例）

コンフィグレーション例

```
<SphSolver>
    <SphUserDefile>
        <Velocity>
            <Param name="u"    value="1.0"/>
            <Param name="v"    value="1.5"/>
            <Param name="w"    value="2.0"/>
        </Velocity>
    </SphUserDefile>
</SphSolver>
```

コード例

```
    // ソルバー要素の取得
    const sphElem* solv_elem = this->getConfigElem();
    const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
    const sphElem* vel_elem = user_elem->getChildElemFirst(2, "Velocity", "Param");
    printf("name = %s, value = %f¥n",
                                vel_elem->getAttrString("name"),
                                vel_elem->getAttrReal4("value"));
    while (vel_elem != NULL) {
        // Solid[@id=500], Solid[@id=255]要素の取得
        vel_elem = user_elem->getChildElemNext(vel_elem, 2, "Velocity", "Param");
        if (vel_elem != NULL) {
            printf("name = %s, value = %f¥n",
                                    vel_elem->getAttrString("name"),
                                    vel_elem->getAttrReal4("value"));
        }
    }
```


４．８　子要素数の取得

int countNumOfElem(const char* elem_name = NULL) const;	
子要素名を指定して子要素数を取得します。	
引数	elem_name　　子要素名
戻り値	子要素数

指定要素の子要素名を指定して見つかった子要素数を取得します
子要素名を省略した場合は、要素名に関係なく子要素数を取得します。

（使用例）

```
// ソルバー要素の取得
const sphElem* solv_elem = this->getConfigElem();
const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
// Velocity 数の取得
int    vel_cnt = user_elem-> countNumOfElem ("Velocity");
```

４．９　子孫要素数の取得

int countNumOfElem(unsigned int num, const char* elem_name00, ...) const;	
子孫要素名を指定して子孫要素数を取得します。	
引数	num　　指定要素名の数 elem_name00　　最初の子要素名 ...　　２番目以降の子要素名
戻り値	子孫要素数

指定要素の子要素名を複数指定し、子孫要素の階層と一致する要素数を取得します。
同一階層、同一親要素でなくとも指定された要素名階層と一致するパターンの要素数を取得します。

（使用例）

コンフィグレーション例

```
<SphSolver>
    <SphUserDefile>
        <Solid id="600" comment="Fe">
            <density   id="1" value="7870.0" />
            <density   id="2" value="7870.0" />
        </Solid>
        <Solid id="500" comment="Al">
            <density   value="2688.0" />
        </Solid>
    </SphUserDefile>
</SphSolver>
```

コード例

```
    // ソルバー要素の取得
    const sphElem* solv_elem = this->getConfigElem();
    const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
    int den_count = user_elem-> countNumOfElem (2, "Solid", "density");
    printf("count = %d ¥n", den_count);
    // 出力
    // count = 3                "Solid/density"パターンの要素は３つ
```

４．１０　要素名、属性値の要素の検索

const sphElem* getChildElemByAttr(const char* elem_name,const char* attr_name,const char* attr_value) const;		
子要素名、属性名、属性値と一致する子要素を取得します。		
引数	elem_name	子要素名
	attr_name	属性名
	attr_value	属性値
戻り値	子要素	

指定要素の子要素名、属性名とその値を指定し一致する要素を取得します。
要素名、属性名、属性値の大文字・小文字は区別しません。

（使用例）

コンフィグレーション例

```
<SphSolver>
    <SphUserDefile>
        <Heat_BC_Face name="coef_of_heat_transfer" value="1.2e-2"/>
    </SphUserDefile>
</SphSolver>
```

コード例

```
// ソルバー要素の取得
const sphElem* solv_elem = this->getConfigElem();
const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
const sphElem* heat_elem = user_elem->getChildElemByAttr(
                                        "Heat_BC_Face",
                                        "name",
                                        "Coef_of_Heat_Transfer");
printf("value = %f¥n", heat_elem->getAttrReal4("value"));
```

４．１１　要素名の取得

const char*getElemName() const;	
要素名を取得します。	
引数	なし
戻り値	要素名

指定要素の要素名を取得します。

（使用例）

```
// ソルバー要素の取得
const sphElem* solv_elem = this->getConfigElem();
printf("elem name = %s¥n", solv_elem->getElemName());
// 出力：elem_name = SphSolver
const sphElem* user_elem = solv_elem->getElem(SPH_CFG_NAME_USERDEFINE);
printf("elem name = %s¥n", user_elem ->getElemName());
// 出力：elem_name = SphUserDefine
```

４．１２　属性値の取得(const char*)

const char*　getAttrString(const char* name) const;	
属性値を char*で取得します。	
引数	name　　属性名
戻り値	属性値(const char*)

属性値を char*で取得します。

４．１３　属性値の取得(short)

short getAttrShort (const char* name) const;	
属性値を short で取得します。	
引数	name　　属性名
戻り値	属性値(short)

属性値を short で取得します。

４．１４　属性値の取得(int)

int getAttrInt　(const char* name) const;	
属性値を int で取得します。	
引数	name　　属性名
戻り値	属性値(int)

属性値を int で取得します。

４．１５　属性値の取得(long)

long getAttrLong　(const char* name) const;	
属性値を long で取得します。	
引数	name　　属性名
戻り値	属性値(long)

属性値を long で取得します。

４．１６　属性値の取得(long long)

long long getAttrLLong (const char* name) const;	
属性値を long long で取得します。	
引数	name　　属性名
戻り値	属性値(long long)

属性値を long long で取得します。

４．１７　属性値の取得(float)

float getAttrReal4 (const char* name) const;	
属性値を float で取得します。	
引数	name　　属性名
戻り値	属性値(float)

属性値を float で取得します。

４．１８　属性値の取得(double)

double getAttrReal8 (const char* name) const;	
属性値を double で取得します。	
引数	name　　属性名
戻り値	属性値(double)

属性値を double で取得します。

４．１９　属性値の取得(SphDCType)

SphDCType getAttrSphDcType (const char* name) const;	
属性値を SphDCType 型で取得します。	
引数	name　　属性名
戻り値	属性値(SphDCType 型)

属性値をデータクラスタイプ値（SphDCType 型）で取得します。

SphDCType 型は"include/data/sphDataDefs.h"で定義されている enum 型です。

属性値の文字列から以下の enum SphDCType 値を返します。

No	属性値	SphDCType 値	データクラスタイプ
1	"scalar1d"	SPH_DC_ARRAY_1D	1D データクラス データ配列{i}
2	"scalar2d"	SPH_DC_ARRAY_2D	2D データクラス データ配列{i, j}
3	"scalar3d"	SPH_DC_ARRAY_3D	3D データクラス データ配列{i, j, k}
4	"vector1d"	SPH_DC_ARRAY_1DN	1D データクラス データ配列{i, l}
5	"vector1drv"	SPH_DC_ARRAY_1DNRV	1D データクラス データ配列{i, l}
6	"vector2d"	SPH_DC_ARRAY_2DN	2D データクラス データ配列{i, j, l}
7	"vector2drv"	SPH_DC_ARRAY_2DNRV	2D データクラス データ配列{l, i, j}
8	"vector3d"	SPH_DC_ARRAY_3DN	3D データクラス データ配列{i, j, k, l}
9	"vector3drv"	SPH_DC_ARRAY_3DNRV	3D データクラス データ配列{l, i, j, k}
10	"unst_node"	SPH_DC_UNST_NODE	非構造データクラス（節点、スカラ）
11	"unst_node_n"	SPH_DC_UNST_NODE_N	非構造データクラス（節点、ベクトル）
12	"unst_node_nrv"	SPH_DC_UNST_NODE_NRV	非構造データクラス（節点、ベクトル、反転）
13	"unst_elem"	SPH_DC_UNST_ELEM	非構造データクラス（要素、スカラ）
14	"unst_elem_n"	SPH_DC_UNST_ELEM_N	非構造データクラス（要素、ベクトル）
15	"unst_elem_nrv"	SPH_DC_UNST_ELEM_NRV	非構造データクラス（要素、ベクトル、反転）

４．２０　属性値の取得(SPL_Datatype)

SPL_Datatype　getAttrSplDataType (const char* name) const;	
属性値を SPL_Datatype 型で取得します。	
引数	name　　属性名
戻り値	属性値(SPL_Datatype 型)

属性値をデータタイプ値（SPL_Datatype 型）で取得します。

SPL_Datatype 型は"include/spl/splmpi_mpi.h"で定義されています。

属性値の文字列から以下の SPL_Datatype 型を返します。

No	属性値	SPL_Datatype 型	データタイプ
1	"char"	SPL_CHAR	char 型データ
2	"unsigned_char"	SPL_UNSIGNED_CHAR	unsigned char データ型
3	"short"	SPL_SHORT	short 型データ型
4	"unsigned_short"	SPL_UNSIGNED_SHORT	unsigned short データ型

5	"int"	SPL_INT	int 型データ型
6	"unsigned_int"	SPL_UNSIGNED	unsigned int データ型
7	"long"	SPL_LONG	long 型データ型
8	"unsigned_long"	SPL_UNSIGNED_LONG	unsigned long データ型
9	"long long"	SPL_LONG_LONG	long long 型データ型
10	"float"	SPL_FLOAT	float 型データ型
11	"double"	SPL_DOUBLE;	double 型データ型

４．２１　属性値の取得(SphCrdDef)

SphCrdDef getAttrSphCrdDef(const char* name) const	
属性値を SphCrdDef 型で取得します。	
引数	name　　属性名
戻り値	属性値(SphCrdDef 型)

属性値をデータ定義位置値（SphCrdDef 型）で取得します。

SphCrdDef 型はは"include/data/sphDataDefs.h"で定義されている enum 型です。

属性値の文字列から以下の SphCrdDef 型を返します。

No	属性値	SphCrdDef 型	データ定義位置
1	"regular"	CRDDEF_REGULAR	レギュラー定義位置
2	"collocate"	CRDDEF_COLLOCATE	コロケート定義位置
3	"staggered_1"	CRDDEF_STAGGERED1	スタッガード１定義位置
4	"staggered_2"	CRDDEF_STAGGERED2	スタッガード２定義位置

４．２２　属性値の取得(bool)

bool isAttrValue　(const char* name,bool default_value = true)const;	
属性値を bool 値で取得します。	
引数	name　　属性名 default_value　　属性が存在しなかった場合の戻り値
戻り値	属性値(bool)

属性値を bool 値で取得します。

属性名で指定された属性が存在しない場合は、default_value の値を返します。

default_value を省略した場合は、true を返します。

属性値の文字列により bool 値を決定します。

No	属性値	戻り値(bool)
1	"true"	true
2	"on"	true
3	"yes"	true
4	"with"	true
5	上記以外	属性が存在する：false
		属性が存在しない：default_value

４．２３　属性の有無

bool existAttribute(const char* name) const;	
属性名が存在するかチェックします。	
引数	name　　属性名
戻り値	true:属性が存在する。

属性名で指定された属性が存在するかチェックします。

５．　V-Sphere 形式コンフィグレーションへのアクセス

SPHERE では、V-Sphere 形式のコンフィグレーションファイルを指定することもできます。

V-Sphere 形式のコンフィグレーションへのアクセスは以下のメソッドを使用してください。

５．１　V-Sphere 形式のコンフィグレーションの取得

(1) V-Sphere 形式のコンフィグレーションクラスの取得

const SklCfg::SklSolverConfig* getSklConfig();	
V-Sphere 形式のコンフィグレーションクラスの取得を行う。	
引数	なし
戻り値	V-Sphere 形式のコンフィグレーション

V-Sphere 形式のコンフィグレーションを取得します。
取得したコンフィグレーションクラスオブジェクトから設定パラメータの取得方法については、"V-Sphere コンフィグレーション取得メソッドマニュアル"を参照してください。

(2) V-Sphere 形式であるかチェック

bool isClassicConfig();	
指定されたコンフィグレーションが V-Sphere 形式であるかチェックします。	
引数	なし
戻り値	true:V-Sphere 形式のコンフィグレーションファイル指定

V-Sphere 形式のコンフィグレーションファイルが起動引数に指定された場合、true を返します。
V-Sphere 形式と Sphere 形式の判断は以下の属性値にて行います。
（Sphere 形式ルート要素）

```
<?xml version="1.0"?>
<SphereConfig version="2.0.0">
    ....
</ SphereConfig>
```

ルート要素"SphereConfig"
"version"属性値＝"2.0.0" : Sphere 形式
"version"属性なし、又は"2.0.0"以外 : V-Sphere 形式