SYBASE®
An SAP® Company

移行ガイド

Sybase Event Stream Processor 5.0

ドキュメント ID： DC01735-01-0500-01
改訂： 2011 年 12 月

目次

第 1 章 Aleri Streaming Platform 3.x へのアップグレード

Aleri Streaming Platform バージョン 2.x を実行している場合、移行を行う前に **esp_upgrade** ユーティリティを使用して、3.x にアップグレードします。

esp_upgrade ユーティリティを使用して Aleri 2.x から Aleri 3.x にアップグレードします。AleriML ファイルを CCL に変換する Event Stream Processor 変換ツールは AleriML バージョン 3.x に基づいています。このため、モデルを CCL に移行する前に Aleri 2.x 用に記述されたモデルを Aleri 3.x 用に変換します。

このユーティリティによって指定された AleriML ファイルからプロジェクト・データが読み込まれ、標準の XML 出力ファイルに書き込まれます。ユーティリティはほとんどのアップグレードの問題を自動的に処理しますが、以下に関しては手動での更新が必要になります。

- ルールの変換 (**esp_upgrade** を使用して XML ファイル内でコメントアウト)
- 式の変換 (ルールの変換でのみ実行されるため、同様に XML ファイル内でコメントアウト)
- ローのローカル記憶領域の eventCaches への変換

このアップグレードは次の手順で行います。

1. コマンド・ラインで以下を実行します。
   ```
   esp_upgrade sourcefile.xml > destinationfile.xml
   ```
   ここで、`sourcefile.xml` は 2.x のプロジェクト・ファイルで、`destinationfile.xml` はアップグレード後の 3.x のファイルです。
2. 必要に応じて出力 XML ファイル (`destination.xml`) を変更し、手動でのアップグレードの問題に対応します。
3. ファイルを保存して閉じます。
   これで、Sybase® Event Stream Processor 移行ツールを実行して AleriML ファイルを CCL に移行する準備ができました。

第 2 章 AleriML モデルの CCL プロジェクトへの移行

esp_aml2ccl ユーティリティを使用して既存の AleriML モデルを CCL プロジェクトに移行します。

正常に移行を完了させるために、CCL 言語に関する基本事項を理解します。変換を実行する前に、『CCL プログラマーズ・ガイド』を参照してください。

AleriML の既存のモデルを CCL に変換するには、次のようにします。

1. 以下のコマンドを実行します。
   esp_aml2ccl -f <filepath>.xml -p <ProjectName> -l<ProjectLocation>
   ここで、**-f <filepath>.xml** は AleriML モデル・ファイルのパスで、**-p <ProjectName>** はターゲットの CCL プロジェクトの名前 (デフォルトは `xml` 入力ファイルのプレフィクス) で、**-l <ProjectLocation>** はプロジェクトの場所の絶対パス (デフォルトは現在の場所) です。
2. 変換によって作成される 3 つのファイルを参照します。
   - <projectName>.ccp - プロジェクトの詳細を含みます。
   - <projectName>.ccl - 変換後の CCL 要素を含みます。
   - <projectName>.err - 変換に関する情報、および変換中に生成されたエラーと警告を含みます。

   注意： [File] メニューから [Convert Aleri Model] を選択して、ESP スタジオから同様の操作を行うこともできます。

参照：
- *第 8 章、「既知の制限事項」* (35 ページ)

変換レポート・ファイル

変換レポート (`.err`) ファイルには、変換手順に関する情報、および AleriML の CCL への変換中に発生したエラーが含まれます。

変換レポート・ファイルには、それぞれに重大度が割り当てられたログ文があります。

重大度	説明
INFO	各移行手順と全体的なプロセスの概要です。例： `INFO:Initialization start for Store element store1.`
WARNING	AleriML モデル・ファイルの移行が完了しておらず、CCL ファイルまたはプロジェクトの配備において手動更新が必要です。例： `The restriction access mechanism in ccl has been changed.It needs to be configured manually.`
ERROR	AleriML モデル・ファイルはエラーを含みます。例： `ExtendStream ExtendStream1 could not be converted to ccl as it could not be initialized.`

第 3 章 要素の移行

Aleri 要素を CCL に移行する方法の例を確認します。

StartUp 要素の移行

CCL では、Aleri の StartUp 要素は **ADAPTER START** 文に移行し、接続グループ参照を使用します。**ATTACH ADAPTER** 文のアダプタに接続グループを割り当てることができます。

AleriML：

```
<StartUp comments="StartUp">
    <ConnectionGroup id="ConnectionGroup1" type="start">
      <ConnectionRef connection=" Connection1"/>
    </ConnectionGroup>
    <ConnectionGroup id="ConnectionGroup2" type="nostart">
      <ConnectionRef connection="Connection2"/>
      </ConnectionGroup>
  </StartUp>
```

CCL：

```
ADAPTER START  GROUPS
ConnectionGroup1 , ConnectionGroup2   NOSTART ;
```

DataLocation 要素の移行

Aleri の DataLocation 要素は CCL に直接移行しません。これは、CCL に同等の要素がないためです。ただし、**ATTACH ADAPTER** 文の CCL アダプタ・プロパティとしてすべての Aleri の DataLocation プロパティを定義できます。

AleriML：

```
<DataLocation id="xml_file_input" type="xml_in">
      <LocationParam name="dir" value="C:/test"/>
      <LocationParam name="matchStreamName" value="false"/>
      <LocationParam name="filePattern" value="*.xml"/>
      <LocationParam name="file" value="test.xml"/>
    </DataLocation>
```

対応する CCL はありません。

ストア要素

AleriML から CCL にストア要素を移行する方法を確認します。

AleriML には、ステートレス、メモリ、ログの 3 つのタイプのストアがあります。CCL にあるのは、メモリとログの 2 つのタイプのみです。CCL では、Aleri のメモリ・ストアは **CREATE MEMORY STORE** 文に移行し、Aleri のログ・ストアは **CREATE LOG STORE** 文に移行します。

例：メモリ・ストアの移行

CCL では、Aleri のメモリ・ストア要素は **CREATE MEMORY STORE** 文に移行します。デフォルトでは、indextype と indexsizehint のプロパティが設定されています。

Aleri の index プロパティは、CCL では、indextype プロパティに移行します。AleriML は `index="{tree|hash|list}"` を持つのに対し、CCL は `INDEXTYPE={'tree'|'hash'}` を持ちます。デフォルトでは、AleriML の `index="list"` は、CCL では `INDEXTYPE='tree'` に移行します。

AleriML：

```
<Store file="store1" id="store1" kind="memory" index="tree"/>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
```

例：ログ・ストアの移行

CCL では、Aleri のログ・ストア要素は **CREATE LOG STORE** 文に移行します。デフォルトでは、reservepct、indexsizehint、sync、ckcount のプロパティが設定されています。

AleriML：

```
<Store id="events" kind="log" fullsize="1024" file="store/events" />
```

CCL：

```
CREATE  LOG  STORE events PROPERTIES  FILENAME ='store/events',
MAXFILESIZE =1024,  RESERVEPCT =20,  SYNC = false ,  CKCOUNT =10000,
INDEXSIZEHINT =8;
```

InConnection 要素の移行

CCL では、Aleri の InConnection 要素は **ATTACH INPUT ADAPTER** 文に移行し、アダプタ・プロパティとして InConnection と DataLocation のプロパティが定義されます。Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、SourceStream 要素は **CREATE INPUT WINDOW** 文に移行します。

AleriML：

```
<Store file="store1" id="store1" kind="memory"/>
   <DataLocation id="xml_file_input" type="xml_in">
      <LocationParam name="dir" value="C:/test"/>
      <LocationParam name="matchStreamName" value="false"/>
      <LocationParam name="filePattern" value="*.xml"/>
       </DataLocation>
<SourceStream id="alldatatypes" store="store1">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
    <InConnection location="xml_file_input" name="InConn1">
         <ConnectionParam name="file" value="test.xml"/>
    </InConnection>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1;
ATTACH INPUT ADAPTER InConn1
 TYPE xml_in
 TO alldatatypes
 PROPERTIES
dir='C:/test',
file='test.xml',
filePattern='*.xml',
matchStreamName=false;
```

OutConnection 要素の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、SourceStream 要素は **CREATE INPUT WINDOW** 文に移行し、OutConnection 要素は **ATTACH OUTPUT ADAPTER** 文に移行します。

Aleri の OutConnection と DataLocation のプロパティは、アダプタ・プロパティとして定義されます。

AleriML：

```
   <Store file="store1 id="store1" kind="memory"/>
      <DataLocation id="xmlOut" type="xml_out"></DataLocation>
    <SourceStream id="alldatatypes" store="store1">
         <OutConnection location="xmlOut" name=" OutConn1">
            <ConnectionParam name="dir" value="output"/>
            <ConnectionParam name="file" value="testOut.xml"/>
            <ConnectionParam name="outputBase" value="true"/>
      </OutConnection>
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1;
ATTACH OUTPUT ADAPTER OutConn1
 TYPE xml_out
 TO alldatatypes
 PROPERTIES
dir='output',
file='testOut.xml',
outputBase=true;
```

SourceStream 要素

さまざまなケースの SourceStream 要素を AleriML から CCL に移行する方法を確認します。

例：ステートフル・ストアを持つ SourceStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、SourceStream 要素は入力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8 ;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a )
 STORE store1;
```

例：insertOnly 属性を持つ SourceStream の移行

Aleri のストア要素は **CREATE MEMORY STORE** 文に移行します。CCL では、ストリームはステートレスであり INSERT opcode のみをサポートします。ウィンドウはステートフルでありすべての opcode をサポートします。

したがって、この例は、出力ウィンドウが続く入力ストリームに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1"
insertOnly="true">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  STREAM Ccl_0_alldatatypes
```

```
SCHEMA (id INTEGER, a LONG, charData STRING);
CREATE  OUTPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
 PRIMARY KEY DEDUCED
 STORE store1
 AS
 SELECT Ccl_0_alldatatypes.id, Ccl_0_alldatatypes.a,
Ccl_0_alldatatypes.charData FROM Ccl_0_alldatatypes GROUP BY
Ccl_0_alldatatypes.id, Ccl_0_alldatatypes.a;
```

例：FilterExpression を持つ SourceStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、FilterExpression は **WHERE** 句に移行します。

したがって、この例は、**WHERE** 句を持つ出力ウィンドウが続く、入力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
    <FilterExpression>alldatatypes.charData in ('aa','bb','cc')</
FilterExpression>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW Ccl_0_alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1;
CREATE  OUTPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
 SELECT  *  FROM Ccl_0_alldatatypes
WHERE Ccl_0_alldatatypes.charData in ('aa','bb','cc');
```

例：autogen 属性を持つ SourceStream の移行

CCL では、nextval() は Aleri autogen 属性と同等であり、Aleri のストア属性は **CREATE MEMORY STORE** 文に移行します。

したがって、この例は、出力ウィンドウが続く入力ウィンドウに移行します。また、**SELECT** 句の値として設定された nextval() を持つ出力ウィンドウ・スキーマ用に ID カラムが含まれています。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1"
insertOnly="true">
    <Column datatype="int64" key="true" name="id" autogen="true"/>
    <Column datatype="int64" key="false" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  STREAM Ccl_0_alldatatypes
SCHEMA (a LONG, charData STRING);
CREATE  OUTPUT  WINDOW alldatatypes
SCHEMA (id LONG, a LONG, charData STRING)
PRIMARY KEY (id)
 STORE store1
 AS
 SELECT  nextval() AS id, Ccl_0_alldatatypes.a AS a,
Ccl_0_alldatatypes.charData AS charData FROM Ccl_0_alldatatypes;
```

例：InputWindow を持つ SourceStream の移行

CCL では、Aleri の InputWindow 要素は **KEEP** 句にマップします。

したがって、この例は、**KEEP** 句を持つ入力ウィンドウに移行します。

AleriML：

```
  <Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1" >
   <InputWindow type="records" value="1000" slack="500"/>
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
KEEP 1000 ROWS  SLACK 500;
```

例：ステートレス・ストアを持つ SourceStream の移行

CCL では、アタッチされたステートレス・ストアを持つ Aleri の SourceStream 要素は入力ストリーム要素に移行します。これは、CCL ではストリームはステートレスであるためです。

AleriML：

```
   <Store file="store1" fullsize="64" id="store1" kind="stateless"/>
    <SourceStream id="alldatatypes" store="store1"
insertOnly="true">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
```

CCL：

```
CREATE  INPUT  STREAM alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING);
```

CopyStream 要素

さまざまなケースの CopyStream 要素を AleriML から CCL に移行する方法を確認します。

例：ステートフル・ストアを持つ CopyStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、CopyStream 要素は出力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<CopyStream id="copydatatypes" store="store1"
istream="alldatatypes"/>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW copydatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
SELECT * FROM alldatatypes;
```

例：InputWindow を持つ CopyStream の移行

この例では、Aleri の CopyStream 要素は、アタッチされたステートフル (メモリ) ストアと、入力ストリーム "alldatatypes" の InputWindow を持ちます。CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、InputWindow 要素は **KEEP** 句にマップします。

したがって、この例は、出力ウィンドウが続く、**KEEP** 句を持つローカル・ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <CopyStream id="copydatatypes" store="store1"
istream="alldatatypes">
    <InputWindow stream="alldatatypes" type="records" value="1000"
slack="500"/>
    </CopyStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE LOCAL  WINDOW Ccl_1_alldatatypes SCHEMA (id INTEGER, a LONG,
charData STRING)
PRIMARY KEY (id, a)
 STORE store1
KEEP 1000 ROWS  SLACK 500
 AS
 SELECT alldatatypes.id AS id, alldatatypes.a AS a,
alldatatypes.charData AS charData FROM alldatatypes;
CREATE  OUTPUT  WINDOW copydatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
SELECT * FROM Ccl_1_alldatatypes;
```

ステートフル・ストアを持つ UnionStream の移行

この例では、Aleri の UnionStream 要素は、アタッチされたステートフル (メモリ) ストアと、ユニオン演算に含める 2 つの入力ストリーム ("alldatatypes" と "alldatatypes1") を持ちます。CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、UnionStream 要素は **UNION** 句を持つ出力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<UnionStream id="uniondatatypes" istream="alldatatypes
alldatatypes1" store="store1">
</UnionStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW uniondatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
 SELECT * FROM alldatatypes
 UNION
 SELECT * FROM alldatatypes1;
```

FilterStream 要素

さまざまなケースの FilterStream 要素を AleriML から CCL に移行する方法を確認します。

例：ステートフル・ストアを持つ FilterStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、Aleri の FilterStream 要素は出力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<FilterStream id="filterdatatypes" store="store1"
istream="alldatatypes">
 <FilterExpression>alldatatypes.charData in ('aa','bb','cc')</
FilterExpression>
</FilterStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW filterdatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
SELECT  * FROM alldatatypes
WHERE alldatatypes.charData in ('aa','bb','cc');
```

例：InputWindow を持つ FilterStream の移行

この例では、AleriML の FilterStream は、アタッチされたステートフル・ストアと、入力ストリーム "alldatatypes" の InputWindow を持ちます。CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、InputWindow 要素は **KEEP** 句にマップします。

したがって、この例は、**WHERE** 句を持つ出力ウィンドウが続く、**KEEP** 句を持つローカル・ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<FilterStream id="filterdatatypes" store="store1"
istream="alldatatypes">
 <InputWindow stream="alldatatypes" type="records" value="1000"
slack="500"/>
 <FilterExpression>alldatatypes.charData in ('aa','bb','cc')</
FilterExpression>
</FilterStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
 CREATE LOCAL  WINDOW Ccl_1_alldatatypes SCHEMA (id INTEGER, a LONG,
charData STRING)
PRIMARY KEY (id, a)
 STORE store1
KEEP 1000 ROWS  SLACK 500
 AS
 SELECT alldatatypes.id AS id, alldatatypes.a AS a,
alldatatypes.charData AS charData FROM alldatatypes;

CREATE  OUTPUT  WINDOW filterdatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
SELECT * FROM Ccl_1_alldatatypes
WHERE Ccl_1_alldatatypes.charData in ('aa','bb','cc');
```

ステートフル・ストアを持つ ComputeStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、ComputeStream 要素は出力ウィンドウに移行します。ColumnExpression 要素の名前属性の値は、出力ウィンドウの **SELECT** 句のカラムの名前となります。ColumnExpression 要素の値は、**SELECT** 句の値となります。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<ComputeStream id="Normalizeddatatypes" store="store1"
istream="alldatatypes">
    <ColumnExpression key="true" name="id1" >alldatatypes.id</
ColumnExpression>
    <ColumnExpression name="a1">alldatatypes.a * 5</
ColumnExpression>
    <ColumnExpression name="charData1">alldatatypes.charData</
ColumnExpression>
</ComputeStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW Normalizeddatatypes
SCHEMA (id 1 INTEGER, a1 LONG, charData1 STRING)
PRIMARY KEY (id)
 STORE store1
 AS
SELECT alldatatypes.id AS id1, alldatatypes.a * 5 AS a1,
alldatatypes.charData AS charData1
FROM alldatatypes;
```

ステートフル・ストアを持つ **ExtendStream** の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行します。Aleri の ExtendStream 要素が入力ストリームからのすべてのカラムを拡張する場合、CCL では、この要素は入力ウィンドウからの拡張カラムを持つ出力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<ExtendStream id="Extenddatatypes" store="store1"
istream="alldatatypes">
<ColumnExpression name="a1">alldatatypes.a * 109.0</
ColumnExpression>
<ColumnExpression
name="charData">concat(alldatatypes.charData,'test')</
ColumnExpression>
</ExtendStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW Extenddatatypes
SCHEMA (id INTEGER, a LONG, charData STRING, a1 FLOAT)
PRIMARY KEY (id, a)
 STORE store1
```

```
 AS
SELECT alldatatypes.id AS id, alldatatypes.a AS a,
concat(alldatatypes.charData,'test') AS charData, alldatatypes.a *
109.0 AS a1 FROM alldatatypes;
```

ステートフル・ストアを持つ AggregateStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、AggregateStream 要素は出力ウィンドウに移行します。ColumnExpression 要素の名前属性の値は、出力ウィンドウの **SELECT** 句のカラムの名前となります。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
<AggregateStream id="Aggrdatatypes" store="store1"
istream="alldatatypes">
<ColumnExpression key="true" name="id">alldatatypes.id</
ColumnExpression>
<ColumnExpression key="false" name="maxA">max(alldatatypes.a)</
ColumnExpression>
</AggregateStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  OUTPUT  WINDOW Aggrdatatypes
SCHEMA (id INTEGER, maxA LONG)
 PRIMARY KEY DEDUCED
 STORE store1
 AS
SELECT alldatatypes.id AS id, max(alldatatypes.a) AS maxA
FROM alldatatypes
GROUP BY alldatatypes.id;
```

JoinStream 要素

さまざまなケースの JoinStream 要素を AleriML から CCL に移行する方法を確認します。

Event Stream Processor でのジョインの使用と Aleri Streaming Platform でのジョインの使用は、いくつかの点で異なります。

Aleri Streaming Platform	Event Stream Processor
挿入専用のストリームのジョインを行う場合にのみ内部ジョインをサポートします。	内部ジョインの制限はありません。

Aleri Streaming Platform	Event Stream Processor
多対多はサポートしません。	多対多をサポートします。
保存内容の間接的な指定をサポートします。ジョイン・ソースからの保存内容を持つ CopyStream を作成します。このコピーをジョインのソースとして使用します。	メモリ・ストアのジョイン・ソースのみで保存内容の直接的な指定をサポートします。 **注意：** ログ・ストアでは、ジョイン・ソースのコピーで保存内容を作成します。このコピーをジョインのソースとして使用します。
1 つのスレッドで実行されるストリームのみをジョインします。	ジョインの複雑さやタイプに応じて複数のスレッドを作成できます。
ストリーム (ステートレス・ストア要素) とウィンドウ (メモリ・ストア要素またはログ・ストア要素) のジョインはサポートしません。	ストリーム (ステートレス・ストア要素) とウィンドウのジョインをサポートします。レコードがストリームに到達するとこのジョインが実行され、ストリームがジョインのトリガとして機能します。
プライマリ・キーの抽出はサポートしません。	プライマリ・キーを抽出できます。
1 つの文での複数の操作の指定はサポートしません。各操作は個別に定義されます。	1 つの文での複数のジョイン、フィルタ、集約の指定をサポートします。これは、コンパイラにより複数のスレッドに分割されます。

制限事項
Aleri Streaming Platform は、ジョインにおいてターゲットのキー・フィールドが有効であるかどうかの検証を行いません。null 値および不正な挿入、更新、削除を生成したジョインは暗黙的に無視されます。Event Stream Processor のコンパイラは、ジョイン内でキーとして選択できるものを厳密に実施します。

注意： 『CCL プログラマーズ・ガイド』の「キー・フィールド・ルール」を参照してください。

このため、AleriML のキー・ルールに従っていないジョインでは、CCL に変換された際のコンパイルが行われません。コンパイルを行うには、変換された CCL を編集してキー・フィールドを修正してください。

参照：
- *第 8 章、「既知の制限事項」* (35 ページ)

例：Inner ジョインを持つ JoinStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、JoinStream 要素は JOIN 式を持つ出力ウィンドウに移行します。

この例では、AleriML の JoinStream "EqJoindatatypes" は、入力ストリーム "alldatatypes" と "alldatatypes1" 間に内部ジョインを持ち、それらの間に 1 対 1 のマッピングが実装されています。これらの SourceStream 要素 (alldatatypes と alldatatypes1) は、CCL では入力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1">
        <Column datatype="int32" key="true" name="id"/>
  <Column datatype="int64" key="true" name="a"/>
  <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
     <SourceStream id="alldatatypes1" store="store1">
        <Column datatype="int32" key="true" name="id"/>
        <Column datatype="int64" key="true" name="a"/>
        <Column datatype="string" key="false" name="charData1"/>
   </SourceStream>
 <JoinStream id="EqJoindatatypes" istream="alldatatypes
alldatatypes1" store="store1">
 <Join constraints="id=id a=a" table1="alldatatypes"
table2="alldatatypes1" type="inner"/>
<ColumnExpression key="true" name="id">alldatatypes.id</
ColumnExpression>
 <ColumnExpression key="true" name="a">alldatatypes.a</
ColumnExpression>
<ColumnExpression key="false"
name="chardata">alldatatypes.charData</ColumnExpression>
<ColumnExpression key="false"
name="chardata1">alldatatypes1.charData1</ColumnExpression>
</JoinStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
 STORE store1;
CREATE  INPUT  WINDOW alldatatypes1
SCHEMA (id INTEGER, a LONG, charData1 STRING)
PRIMARY KEY (id, a)
 STORE store1;
CREATE  OUTPUT  WINDOW EqJoindatatypes
SCHEMA (id INTEGER, a LONG, chardata STRING, chardata1 STRING)
PRIMARY KEY (id, a)
 STORE store1
```

```
 AS
SELECT alldatatypes.id AS id, alldatatypes.a AS a,
alldatatypes.charData AS chardata, alldatatypes1.charData1 AS
chardata1 FROM
alldatatypes
 INNER JOIN
alldatatypes1
 ON alldatatypes.id = alldatatypes1.id AND alldatatypes.a =
alldatatypes1.a;
```

例：LeftOuter ジョインを持つ JoinStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、JoinStream 要素は JOIN 式を持つ出力ウィンドウに移行します。

Aleri の JoinStream "leftOuterJoindatatypes" は、入力ストリーム "alldatatypes" と "alldatatypes1" 間に左外部ジョインを持ち、それらの間に 1 対多のマッピングが実装されています。これらの SourceStream 要素 (alldatatypes と alldatatypes1) は、CCL では入力ウィンドウに移行します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1">
        <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
     <SourceStream id="alldatatypes1" store="store1">
        <Column datatype="int32" key="true" name="id"/>
        <Column datatype="int64" key="false" name="a1"/>
        <Column datatype="string" key="false" name="charData1"/>
   </SourceStream>
  <JoinStream id="leftOuterJoindatatypes" istream="alldatatypes
alldatatypes1" store="store1">
    <Join constraints="id=id a=a1" table1="alldatatypes"
table2="alldatatypes1" type="leftouter"/>
   <ColumnExpression key="true" name="id">alldatatypes.id</
ColumnExpression>
    <ColumnExpression key="true" name="a">alldatatypes.a</
ColumnExpression>
    <ColumnExpression key="false"
name="chardata">alldatatypes.charData</ColumnExpression>
    <ColumnExpression key="false"
name="chardata1">alldatatypes1.charData1</ColumnExpression>
    </JoinStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id, a)
```

```
 STORE store1;
CREATE  INPUT  WINDOW alldatatypes1
SCHEMA (id INTEGER, a1 LONG, charData1 STRING)
PRIMARY KEY (id)
 STORE store1;
CREATE  OUTPUT  WINDOW leftOuerJoindatatypes
SCHEMA (id INTEGER, a LONG, chardata STRING, chardata1 STRING)
PRIMARY KEY (id, a)
 STORE store1
 AS
SELECT alldatatypes.id AS id, alldatatypes.a AS a,
alldatatypes.charData AS chardata, alldatatypes1.charData1 AS
chardata1 FROM
alldatatypes
 LEFT JOIN
alldatatypes1
 ON alldatatypes.id = alldatatypes1.id AND alldatatypes.a =
alldatatypes1.a1;
```

例：FullOuter ジョインを持つ JoinStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、JoinStream 要素は JOIN 式を持つ出力ウィンドウに移行します。

Aleri の JoinStream "FOuterJoindatatypes" は、入力ストリーム "alldatatypes" と "alldatatypes1" 間に左外部ジョインを持ち、それらの間に 1 対 1 のマッピングが実装されています。CCL では、Aleri の SourceStream 要素 (alldatatypes と alldatatypes1) は、入力ウィンドウに移行します。

ただし、CCL はコンパイルされない場合があります。これは、Event Stream Processor コンパイラは Aleri コンパイラより厳密であるためです。このため、移行ツールで処理できない **SELECT** 句で使用される式にはいくつか変更点があります。これらの変更点を手動で実行します。

1. 関連するストリームとウィンドウの間に 1 対 1 のマッピングが存在するようにします。
2. プライマリ・キー用に `firstnonnull()` を追加します。

AleriML：

```
<Store file="store1" fullsize="64" id="store1" kind="memory"/>
    <SourceStream id="alldatatypes" store="store1">
        <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="false" name="a"/>
    <Column datatype="string" key="false" name="charData"/>
  </SourceStream>
     <SourceStream id="alldatatypes1" store="store1">
        <Column datatype="int32" key="true" name="id"/>
        <Column datatype="int64" key="false" name="a"/>
        <Column datatype="string" key="false" name="charData1"/>
   </SourceStream>
  <JoinStream id="FOuterJoindatatypes" istream="alldatatypes
alldatatypes1" store="store1">
```

```
   <Join constraints="id=id" table1="alldatatypes"
table2="alldatatypes1" type="fullouter"/>
   <ColumnExpression key="true" name="id">alldatatypes.id</
ColumnExpression>
    <ColumnExpression key="false" name="a">alldatatypes1.a</
ColumnExpression>
    <ColumnExpression key="false"
name="chardata">alldatatypes.charData</ColumnExpression>
   <ColumnExpression key="false"
name="chardata1">alldatatypes1.charData1</ColumnExpression>
    </JoinStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE  INPUT  WINDOW alldatatypes
SCHEMA (id INTEGER, a LONG, charData STRING)
PRIMARY KEY (id)
 STORE store1;
CREATE  INPUT  WINDOW alldatatypes1
SCHEMA (id INTEGER, a LONG, charData1 STRING)
PRIMARY KEY (id)
 STORE store1;
CREATE  OUTPUT  WINDOW FOuterJoindatatypes
SCHEMA (id INTEGER, a LONG, chardata STRING, chardata1 STRING)
PRIMARY KEY (id)
 STORE store1
 AS
SELECT firstnonnull(alldatatypes.id) AS id, alldatatypes1.a AS a,
alldatatypes.charData AS chardata, alldatatypes1.charData1 AS
chardata1 FROM
alldatatypes
 FULL JOIN
alldatatypes1
 ON alldatatypes.id = alldatatypes1.id;
```

Global の移行

CCL では、Aleri の Global 要素は **DECLARE END** 文に移行します。

AleriML：

```
<Global>
int32 depth_of_book := 10;
double change_currency(double val) { return val * 1.57; }
</Global>
```

CCL：

```
DECLARE
INTEGER depth_of_book := 10;
FLOAT change_currency(FLOAT val) { return val * 1.57; }
END;
```

FlexStream の移行

CCL では、Aleri のストア要素は **CREATE MEMORY STORE** 文に移行し、FlexStream "compute" はフレックス要素 "Ccl_1_compute" に移行して、出力ウィンドウ "compute" としての OUT、および AleriML で以前に定義されたスキーマが付加されます。メソッド要素は **ON** 句に移行します。

AleriML：

```
<FlexStream id="compute" istream="alldatatypes" store="store1">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="int64" key="true" name="a1"/>
    <Column datatype="string" key="false" name="charData1"/>

   <Method name="inputMethod" stream="alldatatypes">{
        [int32 id; int64 a; | string charData ] record :=
alldatatypes;
        record.a := record.a + 9;
       record.charData := concat(record.charData,'aa');
        output record;
    }</Method>
    </FlexStream>
```

CCL：

```
CREATE FLEX Ccl_1_compute
 IN alldatatypes
 OUT  OUTPUT   WINDOW compute SCHEMA (id INTEGER, a1 LONG, charData1
STRING)
PRIMARY KEY (id, a1)
 STORE store1
BEGIN
ON alldatatypes {
{
        [INTEGER id; LONG a; | STRING charData ] record :=
alldatatypes;
        record.a := record.a + 9;
        record.charData := concat(record.charData,'aa');
        output record;
 }
 };
END;
```

PatternStream の移行

CCL では、Aleri の PatternStream "compute" は出力ストリーム "Ccl_1_Pattern" に移行します。

この出力ストリームでは、**MATCHING** 句と **ON** 句を使用して AleriML からのパターンを定義します。次に、フレックス要素 "Ccl_2_compute" で出力ストリーム "Ccl_1_Pattern" からの入力を取得し、出力ウィンドウ "compute" に出力します。パターン一致の SPLASH コードは、フレックス要素 "Ccl_2_compute" の **ON** 句に移行します。

AleriML：

```
<PatternStream id="compute" istream="alldatatypes" store="store1">
    <Column datatype="int32" key="true" name="id"/>
    <Column datatype="string" key="true" name="charData1"/>
    <Column datatype="string" key="false" name="charData2"/>
    <Local>    int32 idloc := 0;    </Local>
    <Pattern>
        within 1 seconds
        from alldatatypes[charData='aaa'; a=p] as d1,
        alldatatypes[charData='bbb'; a=q] as d2
        on d1 fby d2
        {
            idloc := idloc + 1;
            output [id = idloc; | Symbol1='aaa'; Symbol2='bbb'];
        }
</Pattern>
</PatternStream>
```

CCL：

```
CREATE  OUTPUT  STREAM Ccl_1_Pattern
SCHEMA (d1id INTEGER, d1a LONG, d1charData STRING, d2id INTEGER, d2a
LONG, d2charData STRING)
 AS
 SELECT d1.id AS d1id, d1.a AS d1a, d1.charData AS d1charData, d2.id
AS d2id, d2.a AS d2a, d2.charData AS d2charData FROM alldatatypes
d1, alldatatypes  d2
 MATCHING
 [ 1 SECONDS :  (  d1  , d2 )  ]
 ON
d1.charData = 'aaa' AND d2.charData = 'bbb';

 CREATE FLEX Ccl_2_compute
 IN Ccl_1_Pattern, alldatatypes
 OUT  OUTPUT WINDOW compute SCHEMA (id INTEGER, charData1 STRING,
charData2 STRING)
PRIMARY KEY (id, charData1)
 STORE store1
```

```
BEGIN
DECLARE
        INTEGER idloc := 0;
END;
 ON Ccl_1_Pattern
 {
        {
            idloc := idloc + 1;
            output [id = idloc; | Symbol1='aaa'; Symbol2='bbb'];
        }
 };
 ON alldatatypes
 {
 };
END;
```

第 4 章 ユーザ定義関数の移行

foreign と foreignJava のユーザ定義関数を AleriML から CCL に移行する方法を確認します。

注意： Aleri では、関数にパラメータの変数を使用することが外部関数によって許可されていますが、Event Stream Processor ではこれを許可していません。

例：Foreign 関数の移行

CCL では、Aleri の ComputeStream "foreignStream" は **CREATE OUTPUT WINDOW** 文に移行し、ストア要素は **CREATE MEMORY STORE** 文に移行します。

この例では、Aleri の ComputeStream 要素は ColumnExpression として呼び出される foreign 関数を持ちます。Aleri の foreign 関数は **CREATE LIBRARY** 文に移行し、ColumnExpression の foreign 関数の参照は Lib_0.intfun( ) に移行します。

```
<Store file="store1" id="store1" kind="memory"/>
 <ComputeStream id="foreignStream" istream="eqInput" store="store1">
        <ColumnExpression key="true" name="a">eqInput.a</
ColumnExpression>
        <ColumnExpression key="false" name="intData">foreign("/opt/
aleriTests/lib/unit/foreign1.so",intfun,int32)</ColumnExpression>
        <ColumnExpression key="false" name="charData">foreign("/opt/
aleriTests/lib/unit/foreign1.so",stringfun,string)</
ColumnExpression>
    </ComputeStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE LIBRARY Lib_0 LANGUAGE C FROM  '/opt/aleriTests/lib/unit/
foreign1.so' (
INTEGER  intfun (  );
);
CREATE LIBRARY Lib_1 LANGUAGE C FROM  '/opt/aleriTests/lib/unit/
foreign1.so' (
STRING  stringfun (  );
);
CREATE  OUTPUT  WINDOW foreignStream
SCHEMA (a INTEGER, intData INTEGER, charData STRING)
PRIMARY KEY (a)
 STORE store1
 AS
SELECT eqInput.a AS a, Lib_0.intfun( ) AS intData, Lib_1.stringfun( )
```

```
AS charData
FROM eqInput;
```

例：ForeignJava 関数の移行

CCL では、ComputeStream "foreignStream" は **CREATE OUTPUT WINDOW** 文に移行し、ストア要素は **CREATE MEMORY STORE** 文に移行します。

この例では、ComputeStream は ColumnExpression として foreignJava 関数を呼び出します。foreignJava 関数は **CREATE LIBRARY** 文に移行し、ColumnExpression の foreignJava 関数の参照は Lib_0.intFunction0( ) に移行します。

AleriML：

```
<Store file="store1" id="store1" kind="memory"/>
  <ComputeStream id="foreignStream" istream="eqInput"
store="store1">
    <ColumnExpression key="true" name="a">eqInput.a</
ColumnExpression>
    <ColumnExpression key="true" name="b">eqInput.b</
ColumnExpression>
    <ColumnExpression key="false"
name="intData0">foreignJava(Functions,intFunction0,'()I')</
ColumnExpression>
    <ColumnExpression key="false"
name="intData1">foreignJava(Functions,intFunction1,'(II)I', 1, 2)</
ColumnExpression>
  </ComputeStream>
```

CCL：

```
CREATE  MEMORY  STORE store1 PROPERTIES  INDEXTYPE ='tree',
INDEXSIZEHINT =8;
CREATE LIBRARY Lib_0 LANGUAGE Java FROM  'Functions' (
INTEGER  intFunction0 (  );
);
CREATE LIBRARY Lib_1 LANGUAGE Java FROM  'Functions' (
INTEGER  intFunction1 ( INTEGER , INTEGER  );
CREATE  OUTPUT  WINDOW foreignStream
SCHEMA (a INTEGER, b STRING, intData0 INTEGER, intData1 INTEGER)
PRIMARY KEY (a, b)
 STORE store1
 AS
SELECT eqInput.a AS a, eqInput.b AS b, Lib_0.intFunction0( ) AS
intData0, Lib_1.intFunction1 (  1, 2) AS intData1
FROM eqInput;
```

第 5 章 用語の変更

Aleri Streaming Platform 3.x から Sybase Event Stream Processor への移行に伴い、いくつか用語が変更されました。

Aleri Streaming Platform の用語	Event Stream Processor の用語	コメント
データ・モデル	プロジェクト	プロジェクトは複数の異なる要素で構成できます。
フレックス・ストリーム	Flex 演算子	Flex 演算子はストリームやウィンドウと同様に動作します。
有効期限	エイジング	レコードは実際には有効期限はありませんが、定義された期間が経過するとフラグが付けられます。
ロー定義	スキーマ	新しい名前です。
`int32` (データ型)	`int`	新しい名前です。
`int64` (データ型)	`long`	新しい名前です。
`double` (データ型)	`float`	新しい名前です。

第 6 章 移行されるユーティリティ

Aleri Streaming Platform から Sybase Event Stream Processor にユーティリティが移行されます。

Aleri ユーティリティ	Event Stream Processor ユーティリティ
sp	**esp_server**
sp_archive	**esp_iqloader**
sp_cli	**esp_client**
sp_cnc	**esp_cnc**
sp_convert	**esp_convert**
sp_encmodel	**esp_encproject**
sp_kdbin	**esp_kdbin**
sp_kdbout	**esp_kdbout**
sp_playback	**esp_playback**
sp_query	**esp_query**
sp_sql2xml	**esp_compiler**
sp_studio	**esp_studio**
sp_subscribe	**esp_subscribe**
sp_upgrade	**esp_upgrade**
sp_upload	**esp_upload**
sp_monitor	**esp_monitor**
sp_ld	廃止
sp_clustermgr	廃止
sp_clustermon	廃止
sp_histexport	廃止
sp_stream2olap	廃止
sp_server	廃止

第 7 章 データ型のマッピング

AleriML データ型は Event Stream Processor (CCL) データ型にマップします。

AleriML データ型	Event Stream Processor データ型
`int32`	`integer`
`int64`	`long`
`double`	`float`
`string`	`string`
`money`	`money(1)...money(15)`
`date (second precision)`	`date (second precision)`
`timestamp` (ミリ秒単位の精度)	`timestamp` (ミリ秒単位の精度)

第 8 章 既知の制限事項

esp_aml2ccl ユーティリティの制限の移行

- **モジュールとクラスタ要素を持つ AleriML –** モジュールとクラスタ要素を持つ AleriML は CCL に移行されません。これは、代わりにインフラストラクチャ要素で表されるためです。

```
<Cluster id="name of cluster">(Node)*</Cluster>
      <Module id="name of module" >{Module | DataLocation | Store
| Stream}*</Module>
```

- **アクセス制限要素 –** AleriML restrictAccess 属性とストリーム要素は CCL に移行されません。これは、代わりにインフラストラクチャ要素で表されるためです。
- **JoinStream の移行 –** ジョインの点では、Event Stream Processor コンパイラは Aleri コンパイラよりも厳密です。ESP サーバでは、移行できてもコンパイルできないケースがいくつか存在します。このようなケースで正常にコンパイルを行うには、以下のように CCL を手動で設定してください。
  - FullOuter ジョインを持つ JoinStream の場合。
    1. 関連するストリームとウィンドウの間に 1 対 1 のマッピングが存在するようにします。
    2. プライマリ・キー用に `firstnonnull()` を追加します。
  - LeftOuter ジョインを持ち、多対多のカーディナリティを持つ JoinStream の場合。
    - 関連するストリームとウィンドウの間に 1 対 1 または多対 1 のマッピングが存在するようにします。
  - Inner ジョインを持つ、1 対 1 のカーディナリティを持たない JoinStream の場合。
    - 関連するストリームとウィンドウの間に 1 対 1 のマッピングが存在するようにします。
- **CCL に移行できない Aleri 要素の属性 –**
  - ストリーム – type、oldid、convdst、ofile、expiryTimeField
  - ストア – oldid
  - SourceStream – convsrc
  - UnionStream – mergeKeys
- **重複カラム –** AleriML では、CCL で許可されない重複カラムを持つ SPLASH コードの重複カラム・レコードが許可されます。**esp_aml2ccl** 移行ツールにより、SPLASH コードがそのまま CCL の SPLASH コードに移行します。SPLASH

レコードに重複カラムが表示される場合は、CCL の手動での更新が必要になる場合があります。

参照：

- *第 2 章、「AleriML モデルの CCL プロジェクトへの移行」* (3 ページ)
- *JoinStream 要素* (17 ページ)

索引

A

C

D

E

F

G

I

J

O

P

S

U

し

す

て

め

ゆ

ろ