Migration Guide

SAP Sybase Event Stream Processor 5.1 SP03

DOCUMENT ID: DC01614-01-0513-01
LAST REVISED: July 2013

Contents

CHAPTER 1 Upgrading to Aleri Streaming Platform 3.x

If you are running version 2.x of Aleri Streaming Platform, use the **esp_upgrade** utility to upgrade to 3.x before attempting your migration.

Use the **esp_upgrade** utility to upgrade from Aleri 2.x to Aleri 3.x. The SAP® Sybase® Event Stream Processor conversion tool, which converts AleriML files to CCL, is based on the 3.x version of AleriML. Therefore, convert any models written for Aleri 2.x to Aleri 3.x before migrating them to CCL.

The utility reads project data from a specified AleriML file and writes it to a standard XML output file. The utility handles most upgrade issues automatically, but requires manual updates for:

- Converting rules (**esp_upgrade** comments these out in the XML file)
- Converting expressions (these occur only within converting rules, so are commented out in the XML file as well)
- Converting row local storage to eventCaches

To perform the upgrade:

1. From a command line, run:
   ```
   esp_upgrade sourcefile.xml > destinationfile.xml
   ```
   where `sourcefile.xml` is the 2.x project file and `destinationfile.xml` is the upgraded 3.x file.
2. Modify the output XML file (`destination.xml`) as necessary to address the manual upgrade issues.
3. Save and close the file.
   You are now ready to run the SAP Sybase Event Stream Processor migration tool to migrate your AleriML files to CCL.

CHAPTER 2 Migrating AleriML Models to CCL Projects

Use the **esp_aml2ccl** utility to migrate existing AleriML models to CCL projects.

For a successful migration overall, obtain a basic understanding of the CCL language. See the *CCL Programmers Guide* before performing any conversions.

Note: Data models containing joins should be recompiled when migrating from ESP 5.0 to any subsequent version of the product. Failure to do so may prevent the ESP Server from loading the compiled data model. In the case that the model does load, the results produced will be incorrect. .

To convert your existing models from AleriML to CCL:

1. Run the command:

   esp_aml2ccl -f <filepath>.xml -p <ProjectName> -l<ProjectLocation>

   where **-f <filepath>.xml** is the path for the AleriML model file, **-p <ProjectName>** is the name of the target CCL project (the default is the prefix of the `xml` input file), and **-l <ProjectLocation>** is an absolute path for the project location (the default is the current location).
2. Refer to the three files created by the conversion.
   - <projectName>.ccp - includes project details.
   - <projectName>.ccl - includes CCL elements after conversion.
   - <projectName>.err - includes information on conversion, as well as any errors and warnings generated during the conversion.

   Note: You can also do this from the ESP Studio by selecting **Convert Aleri Model** from the File menu.

See also

- *Chapter 9, Known Limitations* on page 37

Conversion Report File

The conversion report (`.err`) file contains information on conversion steps, as well as any errors that occur during AleriML to CCL conversions.

The conversion report file contains log statements, each of which is assigned a severity:

Severity	Description
INFO	Introduction for each migration step and overall process. For example: `INFO: Initialization start for Store element store1.`
WARNING	The migration of the AleriML model file is not complete, and requires manual updates, either in the CCL file or project deployment. For example: `The restriction access mechanism in ccl has been changed. It needs to be configured manually.`
ERROR	The AleriML model file includes errors. For example: `ExtendStream ExtendStream1 could not be converted to ccl as it could not be initialized.`

CHAPTER 3 Element Migration

See examples of how the Aleri elements migrate to CCL.

StartUp Element Migration

In CCL, the Aleri StartUp element migrates to an **ADAPTER START** statement, and takes connection group references. You can assign connection groups to adapters in the **ATTACH ADAPTER** statements.

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 Element Migration

The Aleri DataLocation element does not migrate directly to CCL because CCL has no equivalent element. However, you can define all the Aleri DataLocation properties as CCL adapter properties in the **ATTACH ADAPTER** statement.

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>
```

No equivalent CCL.

Store Element

See how the store element migrates from AleriML to CCL.

AleriML has three types of stores: stateless, memory, and log. CCL has only two types: memory and log. In CCL, the Aleri memory store migrates to a **CREATE MEMORY STORE** statement, and the Aleri log store migrates to a **CREATE LOG STORE** statement.

Example: Memory Store Migration

In CCL, the Aleri memory store element migrates to a **CREATE MEMORY STORE** statement. By default, the indextype and indexsizehint properties are set.

The Aleri index property migrates to the indextype property in CCL. AleriML has `index="{tree|hash|list}"` while CCL has `INDEXTYPE={'tree'|'hash'}`. By default, the AleriML `index="list"` migrates to the CCL `INDEXTYPE='tree'`.

AleriML:

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

CCL:

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

Since the list index option is not supported in CCL, you can simulate one in SPLASH using the vector and dictionary data structures. For an example on how to do this, see the *Example: Using a List Store to Preserve Order of Arrival* topic.

Example: Using a List Store to Preserve Order of Arrival

Use vector and dictionary data structures to simulate the Aleri list index option as this option is not supported in CCL.

The CCL example below shows a list store that uses a vector to store the records that need to be accessed in the order of arrival. It also maintains a list index by using a dictionary to quickly update and delete rows in the vector. Similar to a list index in Aleri, deletes are inefficient and must be kept to a minimum because they leave holes in the list.

```
//ListStream is the stream that you need to store as a list so that
you can access the rows in the order of arrival.

CREATE INPUT WINDOW ListStream SCHEMA ( ListKey string , ListUserKey
integer , ListVal integer ) PRIMARY KEY ( ListKey );

//ListUser is the stream that accesses the list in the order of
arrival.

CREATE INPUT WINDOW ListUser SCHEMA ( ListUserKey integer ,
ListUserVal integer ) PRIMARY KEY (ListUserKey);
```

```
//This flex operator serves two purposes. The first purpose is to
maintain the data in the ListStream as a list so that data can be
accessed in the order of arrival. The second purpose is to access the
maintained list whenever a record arrives in the ListUser stream.

CREATE FLEX UserOutput IN ListStream , ListUser OUT OUTPUT WINDOW
UserOutput SCHEMA ( ListUserKey integer , ListUserVal integer ,
First3Total integer ) PRIMARY KEY ( ListUserKey )
BEGIN
    DECLARE
        dictionary( string , integer ) listIdx ;
        vector(typeof(ListStream)) list ;
    END;
    ON ListStream {
          integer opCode := getOpcode(ListStream);
          typeof(ListStream) nullRecord;
          if(opCode = insert) {

//On an insert, put the record into the list and update the index.
            push_back(list, ListStream);
            listIdx[ListStream.ListKey] := size(list) - 1;
        } else if(opCode = update) {

//On an update, replace the existing record with the new one.
            integer idx := listIdx[ListStream.ListKey];
            list[idx] := ListStream;
        } else { //delete

//On a delete, remove the entry. Keep deletes to a minimum because
this leaves a hole in the list.
        integer idx := listIdx[ListStream.ListKey];
            remove(listIdx, ListStream.ListKey);
            list[idx] := nullRecord;
        }
    };
    ON ListUser {
//For every incoming record cycle through the list, get the sum of
values for the first three occurrence of the ListUserKey.
        integer listSize := size(list);
        integer listCounter := 0;

        integer counter := 0;
        integer total := 0;
        typeof(ListStream) rec;

//You cannot use a for loop to iterate over the vector because the
for loop stops whenever there is a null record, and there are nulls
in the vector whenever rows get deleted.
        while(listCounter < listSize) {
            rec := list[listCounter];

            if(rec.ListUserKey = ListUser.ListUserKey){
                total := total + rec.ListVal;
                counter++;
```

```
                if(counter = 3)
                    break;
            }
            listCounter++;
        }
        output setOpcode([ListUserKey=ListUser.ListUserKey;|
UserVal=ListUser.ListUserVal; First3Total=total], upsert);
    };
END;
```

Example: Log Store Migration

In CCL, the Aleri log store element migrates to a **CREATE LOG STORE** statement. By default, the reservepct, indexsizehint, sync, and ckcount properties are set.

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 Element Migration

In CCL, the Aleri InConnection element migrates to an **ATTACH INPUT ADAPTER** statement, and the InConnection and DataLocation properties are defined as adapter properties. The Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the SourceStream element migrates to a **CREATE INPUT WINDOW** statement.

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 Element Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, the SourceStream element migrates to a **CREATE INPUT WINDOW** statement, and the OutConnection element migrates to an **ATTACH OUTPUT ADAPTER** statement.

The Aleri OutConnection and DataLocation properties are defined as adapter properties.

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 Element

See how different cases of the SourceStream element migrate from AleriML to CCL.

Example: SourceStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement and the SourceStream element migrates to an input window.

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;
```

Example: SourceStream with insertOnly Attribute Migration

The Aleri store element migrates to a **CREATE MEMORY STORE** statement. In CCL, streams are stateless and support only the INSERT opcode, and windows are stateful and support all opcodes.

Therefore, this example migrates to an input stream followed by an output window:

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;
```

Example: SourceStream with FilterExpression Migration

In CCL, the Aleri store element migrates to the **CREATE MEMORY STORE** statement and FilterExpression migrates to a **WHERE** clause.

Therefore, this example migrates to an input window followed by an output window with a **WHERE** clause.

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');
```

Example: SourceStream with autogen Attribute Migration

In CCL, the Aleri autogen attribute migrates to the nextval() element, and the Aleri store element migrates to a **CREATE MEMORY STORE** statement.

Therefore, this example migrates to an input window followed by an output window, and the id column is included for the output window schema with nextval() set as the value in the **SELECT** clause.

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;
```

Example: SourceStream with InputWindow Migration

In CCL, the Aleri InputWindow element maps to the **KEEP** clause.

Therefore, this example migrates to an input window with a **KEEP** clause:

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;
```

Example: SourceStream with Stateless Store Migration

In CCL, the Aleri SourceStream element with an attached stateless store migrates to an input stream element because streams are stateless in CCL.

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 Element

See how different cases of the CopyStream element migrate from AleriML to CCL.

Example: CopyStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement and the CopyStream element migrates to an output window.

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;
```

Example: CopyStream with InputWindow Migration

In this example, the Aleri CopyStream element has an attached stateful (memory) store, and an InputWindow for the input stream "alldatatypes". In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement and the InputWindow element maps to a **KEEP** clause.

Therefore, this migrates to a local window with a **KEEP** clause followed by an output window:

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;
```

UnionStream with Stateful Store Migration

In this example, the Aleri UnionStream element has an attached stateful (memory) store and two input streams ("alldatatypes" and "alldatatypes1") that participate in a union operation. In CCL, the Aleri Store element migrates to a **CREATE MEMORY STORE** statement and the UnionStream element migrates to an output window with a **UNION** clause.

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 Element

See how different cases of the FilterStream element migrate from AleriML to CCL.

Example: FilterStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the Aleri FilterStream element migrates to an output window.

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');
```

Example: FilterStream with InputWindow Migration

In this example, the AleriML FilterStream has an attached stateful store and an InputWindow for the input stream "alldatatypes". In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement and the InputWindow element maps to the **KEEP** clause.

Therefore, this example migrates to a local window with a **KEEP** clause, followed by an output window with a **WHERE** clause.

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');
```

ComputeStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement and the ComputeStream element migrates to an output window. The value of the name attribute in the ColumnExpression element becomes the name of the column in the **SELECT** clause of the output window, and the value of the ColumnExpression element becomes the value in the **SELECT** clause.

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;
```

ExtendStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement. Since the Aleri ExtendStream element extends all columns from its input stream, in CCL, it migrates to an output window with extended columns from the input stream.

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;
```

AggregateStream with Stateful Store Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the AggregateStream element migrates to an output window. The value of the name attribute in the ColumnExpression element becomes the name of the column in the **SELECT** clause of the output window.

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 Element

See how different cases of the JoinStream element migrate from AleriML to CCL.

Using joins in Event Stream Processor differs from Aleri Streaming Platform in several ways.

Aleri Streaming Platform	Event Stream Processor
Supports inner joins only when joining insert-only streams.	No restrictions on inner joins.
Does not support MANY-MANY.	Supports MANY-MANY.
Supports specifying retention indirectly. Create a CopyStream with retention from the join source. Use this copy as the source for the join.	Supports specifying retention directly on the join sources for memory stores only. **Note:** For log stores, create a retention on the copy of the join source. Use this copy as the source for the join.
Join streams run in a single thread only.	Multiple threads can be created depending on the complexity and type of the join.
Does not support joining a stream (stateless store element) with a window (memory store or log store element).	Supports joining a stream (stateless store element) with a window. This join is performed when a record arrives on a stream, so that the stream acts as a trigger for the join.
Does not support deducing primary keys.	The primary key can be deduced.
Does not support specifying multiple operations in single statement. Each operation must be defined separately.	Supports specifying joins, filters, and aggregates in a single statement. The compiler breaks this into multiple threads.

Restrictions

Aleri Streaming Platform does not validate whether the key's fields on the target are valid for a join. It silently ignores any joins that produce null values and bad inserts, updates, and deletes. The compiler in Event Stream Processor strictly enforces what can be selected as keys within a

join. As a result, joins that do not follow the key rules in AleriML do not compile when converted to CCL. To compile, you must edit the resulting CCL to correct the key fields. For more details, see *Key Field Rules* in the *CCL Programmers Guide.*

See also

- *Chapter 9, Known Limitations* on page 37

Example: JoinStream with InnerJoin Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the JoinStream element migrates to an output window with JOIN expressions.

In this example, the AleriML JoinStream "EqJoindatatypes" has an inner join between the input streams "alldatatypes" and "alldatatypes1", with One-One mapping between them. These SourceStream elements (alldatatypes and alldatatypes1) migrate to input windows in CCL.

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;
```

Example: JoinStream with LeftOuter Join Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the JoinStream element migrates to an output window with JOIN expressions.

The Aleri JoinStream " leftOuterJoindatatypes" has a left outer join between the input streams "alldatatypes" and "alldatatypes1", with One-Many mapping between these input streams. The SourceStream elements (alldatatypes and alldatatypes1) migrate to an input window in CCL.

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 leftOuterJoindatatypes
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;
```

Example: JoinStream with FullOuter Join Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the JoinStream element migrates to an output window with JOIN expressions.

The Aleri JoinStream "FOuterJoindatatypes" has a left outer join between input streams "alldatatypes" and "alldatatypes1", with One-One mapping between these input streams. In CCL, the Aleri SourceStream elements (alldatatypes and alldatatypes1) migrate to an input window.

However, the CCL may not get compiled because the Event Stream Processor compiler is more strict than the Aleri compiler, and therefore, there are some changes in the expressions used in the **SELECT** clause that cannot be handled by the migration tool. Perform these changes manually.

1. Ensure there is One-One mapping between participating streams and windows.
2. Add `firstnonnull()` for primary keys.

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 Migration

In CCL, the Aleri Global element migrates to a **DECLARE END** statement.

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 Migration

In CCL, the Aleri store element migrates to a **CREATE MEMORY STORE** statement, and the FlexStream "compute" migrates to Flex element "Ccl_1_compute", with OUT as the output window "compute" and with the schema previously defined in AleriML. The method element migrates to an **ON** clause.

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 Migration

In CCL, the Aleri PatternStream "compute" migrates to output stream "Ccl_1_Pattern".

This output stream uses the **MATCHING** and **ON** clauses to define the pattern from AleriML. The Flex element "Ccl_2_compute" then takes input from the output stream "Ccl_1_Pattern" and outputs it to output window "compute". The SPLASH code for pattern matching migrates to the **ON** clause for the Flex element "Ccl_2_compute".

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;
```

CHAPTER 4 User-Defined Functions Migration

See how foreign and foreignJava user-defined functions migrate from AleriML to CCL.

Note: In Aleri, external functions permitted variable numbers of parameters for a function, but Event Stream Processor does not.

Example: Foreign Function Migration

In CCL, the Aleri ComputeStream "foreignStream" migrates to a **CREATE OUTPUT WINDOW** statement, and the Store element migrates to a **CREATE MEMORY STORE** statement.

In this example, the Aleri ComputeStream element has a foreign function that it calls as ColumnExpression. The Aleri foreign function migrates to a **CREATE LIBRARY** statement, and the foreign function references in the ColumnExpression migrate to Lib_0.intfun( ).

```
<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;
```

Example: ForeignJava Function Migration

In CCL, the ComputeStream "foreignStream" migrates to a **CREATE OUTPUT WINDOW** statement, and the Store element migrates to a **CREATE MEMORY STORE** statement.

In this example, the ComputeStream calls a foreignJava function as ColumnExpression. The foreignJava function migrates to a **CREATE LIBRARY** statement, and its references in the ColumnExpression migrate to Lib_0.intFunction0( ).

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;
```

CHAPTER 5 Terminology Changes

There are several changes in terminology from Aleri Streaming Platform 3.x to SAP Sybase Event Stream Processor.

Aleri Streaming Platform Term	Event Stream ProcessorTerm	Comments
Data Model	Project	A project can be composed of a number of different elements.
Flex Stream	Flex Operator	A Flex Operator can behave like a stream or a window.
Expiry	Aging	Records don't actually expire but are flagged after a defined time period.
Row Definition	Schema	New name.
`int32` (datatype)	`int`	New name.
`int64` (datatype)	`long`	New name.
`double`(datatype)	`float`	New name.

CHAPTER 6 Migrated Utilities

Utility migration from the Aleri Streaming Platform to the SAP Sybase Event Stream Processor.

Aleri Utility	Event Stream Processor Utility
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	Deprecated
sp_clustermgr	Deprecated
sp_clustermon	Deprecated
sp_histexport	Deprecated
sp_stream2olap	Deprecated
sp_server	Deprecated

CHAPTER 7 Datatype Mapping

AleriML datatypes map toEvent Stream Processor (CCL) datatypes.

AleriML datatypes	Event Stream Processor datatypes
`int32`	`integer`
`int64`	`long`
`double`	`float`
`string`	`string`
`money`	`money(1)...money(15)`
`date` (second precision)	`date` (second precision)
`timestamp` (millisecond precision)	`timestamp` (millisecond precision)

CHAPTER 8 ESP JDBC Driver

Using Aleri's PostgreSQL JDBC driver, you were able to create an application to connect to the SQL Query port and perform queries on a model. In SAP Sybase Event Stream Processor, you can still use the PostgreSQL JDBC driver, though the process differs somewhat.

SAP Sybase Event Stream Processor provides a thin wrapper over the `postgresql` driver class `org.postgresql.Driver`, which overrides the connection and disconnection mechanisms to connect to projects running in the SAP ESP cluster.

To use the driver, first add the ESP SDK and PostgreSQL JDBC driver jar files (`esp_sdk.jar` and `postgresql.jar`) to the classpath, then register the driver with the Driver Manager as you normally would.

When you have successfully registered the driver, you can use it by retrieving a `Connection` instance from the driver manager by specifying the appropriate JDBC URL, which uses the following format:

```
jdbc:esp://cluster_host:cluster_port/workspace/project[?
arg1=value1&.....]
```

You can specify parameters that are supported by this driver in one of two ways. You can pass them in the `Properties` object when retrieving the connection, or you can specify them in the URL in the standard query parameter format. In either case, the supported parameters and their allowed values are:

- **auth –** The authentication mode to use. If not specified, defaults to user authentication. Allowed values are user and rsa.
- **user –** The user name, when in user authentication mode, or the key alias when in rsa authentication mode.
- **password –** The password for user authentication mode.
- **keystore –** The full path name of a JKS keystore containing the private key to use in rsa authentication mode.
- **storepass –** Password for the keystore.
- **ssl –** Whether to use SSL when connecting. Allowed values are true (connect using SSL) and false.

The following sample passes parameters in the URL in the standard query parameter format:

```
 Class.forName("com.mycompany.esp.jdbc.Driver" );
  String uri = "jdbc:esp://cluster_host:cluster_port/workspace/
project?ssl=true"
  Connection conn = null;
  conn = DriverManager.getConnection(uri);
```

The following sample passes parameters in the properties object:

```
 Class.forName("com.mycompany.esp.jdbc.Driver" );
 Properties props = new Properties();
 props.out("ssl", "true");
 String uri = "jdbc:esp://cluster_host:cluster_port/workspace/
project"
 Connection conn = null;
 conn = DriverManager.getConnection(uri, props);
```

CHAPTER 9 Known Limitations

Migration limitations of the **esp_aml2ccl** utility.

- **AleriML with module and cluster elements –** AleriML with module and cluster elements is not migrated to CCL because it is represented by infrastructure elements instead.

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

- **Restrict access elements –** The AleriML restrictAccess attribute and Stream element are not migrated to CCL because they are represented by infrastructure elements instead.
- **JoinStream migration –** The Event Stream Processor compiler is more strict with joins than the Aleri compiler. Some cases may migrate but may not get compiled on the ESP Server. To successfully compile in such cases, manually configure the CCL for:
  - JoinStream with FullOuter join.
    1. Ensure One-One mapping between participating streams and windows.
    2. Add `firstnonnull()` for primary keys.
  - JoinStream with LeftOuter join with Many-Many cardinality.
    - Ensure One-One or Many-One mapping between participating streams and windows.
  - JoinStream with InnerJoin with no One-One cardinality.
    - Ensure One-One mapping between participating streams and windows.

  Note: Data models containing joins should be recompiled when migrating from ESP 5.0 to any subsequent version of the product. Failure to do so may prevent the ESP Server from loading the compiled data model. In the case that the model does load, the results produced will be incorrect.
- **Aleri element attributes that do not migrate to CCL –**
  - Stream – type, oldid, convdst, ofile, expiryTimeField
  - Store – oldid
  - SourceStream – convsrc
  - Union stream – mergeKeys
- **Duplicate columns –** CCL does not allow SPLASH code duplicate column records. The **esp_aml2ccl** migration tool migrates SPLASH code to CCL SPLASH code as is. You may need to update the CCL manually if duplicate columns are present in the SPLASH record.

See also

- *Chapter 2, Migrating AleriML Models to CCL Projects* on page 3

- *JoinStream Element* on page 18

CHAPTER 10 Deprecated Features

While most features available in Aleri Streaming Platform have been migrated to SAP Sybase Event Stream Processor, some features have not been.

Examples of features that have not been migrated include the ability to make dynamic service modifications, the inbound platform-platform adapter, and the ODBC driver for on-demand SQL queries.

If you are used to working with a particular Aleri Streaming Platform feature and cannot find an equivalent feature in Event Stream Processor, contact Technical Support for details on whether the feature exists in Event Stream Processor under a different name, whether the feature is in the process of being migrated, or whether it has been deprecated.

Index

A

C

D

E

F

G

I

J

K

L

M

O

P

S

T

U