UNISYS

Agile Business Suite

How To Use DataReader in Windows Runtime

Applied to: NET – 2.0.1400 (and higher)

October 2016

How to Use DataReader in Windows Runtime

SUMMARY

This document contains information on how to configure Agile Business Suite (AB Suite™) Windows Runtime to use the MSSQL feature **DataReader** when reading from the database via LDL commands such as Determine, Lookup and ForEach. Note that in the remainder of this document wherever "Determine commands" is referred to, the same information applies to Lookup read loops (for example, Lookup Every) and ForEach commands.

DataReader is a very efficient technique for reading records from an MSSQL Server database. For example, it is more efficient than the standard technique used by AB Suite Windows Runtime – Server-side Cursors. However, it has some limitations which mean it cannot be used in some cases. And it has some behavioral characteristics which mean that extended use of DataReader should be done carefully.

This document describes the way in which you can configure a runtime system to use DataReader selectively i.e. for all or part of your system (for example, in defined ispecs and reports only).

Note that the current implementation for how to define which parts of your system will use DataReader is an interim implementation only. This is provided now so that customers can experiment with the use of DataReader and get the benefits from it now. In the future, the ability to define how DataReader is used by different parts of your system will be done in a different and probably more integrated way.

DataReader versus Cursors

The default technique used by AB Suite Windows Runtime (and EAE) to read records from the database is **Server-side Cursors**. With Cursors, the database will essentially return one record at a time. If a Determine command is executed to read say 100 records, then the database will be accessed 100 times, returning one record on each iteration of the Determine loop.

DataReader works differently. When a Determine loop is entered a separate thread is started (internally within SQL Server) to retrieve records into an internal memory cache. This record retrieval can be done very efficiently with many records being retrieved in blocks. Then, as the Determine code requests the next record during each loop iteration, the record is returned from the memory cache (rather than the physical database). The overall effect is that many records can be retrieved with much less physical database IO than with Cursors, and the Determine code will execute much faster.

It should be clear that DataReader will provide a measurable performance benefit for Determine commands that read multiple records – the more records read the greater the benefit. It follows that DataReader will provide little or no performance benefit if only a few records are read. In fact, there are some situations where DataReader will perform worse than Cursors, due to the overhead of retrieving multiple records into the memory cache that ultimately are not needed (for example, because the code execution breaks out of the Determine loop).

DataReader Limitations

DataReader is not valid for use with a read loop in which a DB Commit is done. This means that DataReader cannot be used for Determine commands where a **Sleep** command occurs anywhere within the loop. This includes situations where a method invoked from within the loop contains a Sleep command.

This is handled automatically by the AB Suite software. It will automatically work out that a specific Determine command contains a Sleep. Therefore, for that Determine command, code will be generated to only use Cursors (i.e. DataReader will not be used).

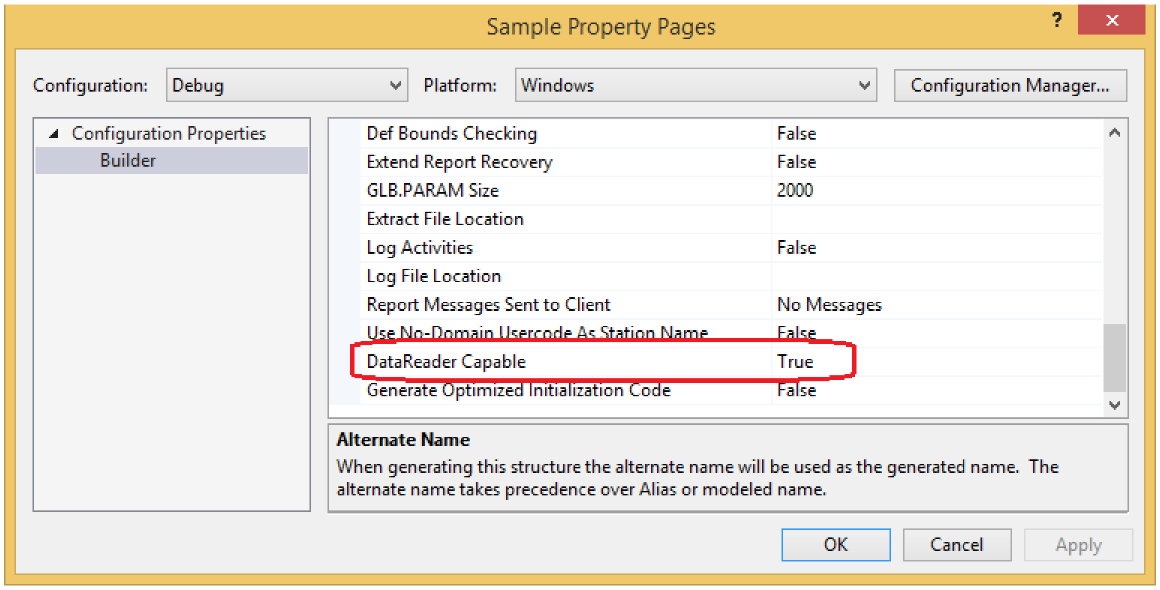
So it is perfectly safe to enable DataReader for parts of the system where you know that Sleep commands are used. The software will automatically work out internally which commands can use DataReader and which cannot. For example, you may have a report that contains say five Determine commands and one of these contains a Sleep. When executed with DataReader enabled, this report will run with DataReader for four of the Determine commands and Cursors for the one Determine that contains a Sleep.

DataReader Code Generation

During the Builder phase, code can be generated for most Determine commands to allow them to be capable of using both Cursors and DataReader at runtime.

By default, only very simple Determine commands are generated to be DataReader-capable. This will be done for Determine loops that do not contain any nested database commands (other Determines, Store(), Flags, etc), Sleep commands, or any method calls.

The generation of DataReader-capable code for many other Determine commands can be enabled by setting a configuration property. This property is available on the Segment object and is called ‘DataReader Capable’. It is set to ‘True’ by default.



Def Bounds Checking	False
Extend Report Recovery	False
GLB.PARAM Size	2000
Extract File Location	
Log Activities	False
Log File Location	
Report Messages Sent to Client	No Messages
Use No-Domain Usercode As Station Name	False
DataReader Capable	True
Generate Optimized Initialization Code	False

Alternate Name
When generating this structure the alternate name will be used as the generated name. The alternate name takes precedence over Alias or modeled name.

This generates code to take advantage of the DataReader feature in SQL Server called "Multiple Active Result Sets" (MARS). This is done for all Determine commands except those that include a Sleep command anywhere within the loop.

This configuration property is available in all AB Suite versions from 5.0. Prior to version 5.0 this DataReader code generation behavior is controlled via a Builder Registry setting called "GenerateCodeForMARS". The following is the definition of this Registry setting:

32-bit Operating System:

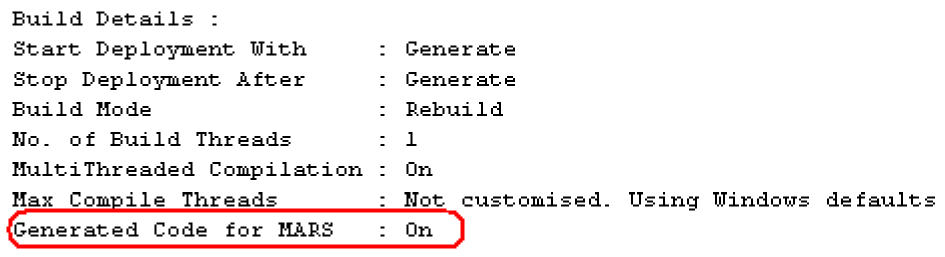
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Unisys\System Modeler\Features\Builder]
"GenerateCodeForMARS"=dword:00000000
```

64-bit Operating System:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Unisys\System Modeler\Features\Builder]
"GenerateCodeForMARS"=dword:00000000
```

The value of this DWORD setting does not matter - it just has to be present.

You can verify that this registry setting has been defined correctly and is active via the Build log from any build operation. The following message will be displayed near the top of the build log when this registry setting has been detected. This indicates that all Determine commands (other than those that contain **Sleep** commands) will be built to be DataReader-capable. If the registry setting has been defined incorrectly (perhaps with the wrong name or in the wrong location in the registry) and is not detected, then this message will not appear at all in the build log.



```
Build Details :
Start Deployment With     : Generate
Stop Deployment After     : Generate
Build Mode                : Rebuild
No. of Build Threads      : 1
MultiThreaded Compilation : On
Max Compile Threads       : Not customised. Using Windows defaults
Generated Code for MARS   : On
```

DataReader Behaviour

It is important to understand that DataReader can cause some Determine command behaviour differences compared to Cursors.

First, it is necessary to understand how Cursors works. A fundamental characteristic of Cursors record retrieval is that it reads one record at a time, with the next record being retrieved from the database at the beginning of each iteration of a Determine loop. If the Profile or Table being read is updated during the read loop such that the content or set of records yet to be returned changes then these changes will be visible to the later iterations. For example, if a new record is stored in the table such that it is included on the Profile being read, then this record will be returned by later iterations of the read loop.

With DataReader this will not work the same. At the beginning of a Determine command, a separate thread will be started to read the complete result set of records into an intermediate memory cache. This result set within the memory cache will not be updated by any changes to the actual records that happens during the read loop. This means that the Determine will continue to return records from the result set in the memory cache even though the physical records in the database may now be different. If this situation exists, then you should use settings that will disable the use of DataReader for this part of the code (see below).

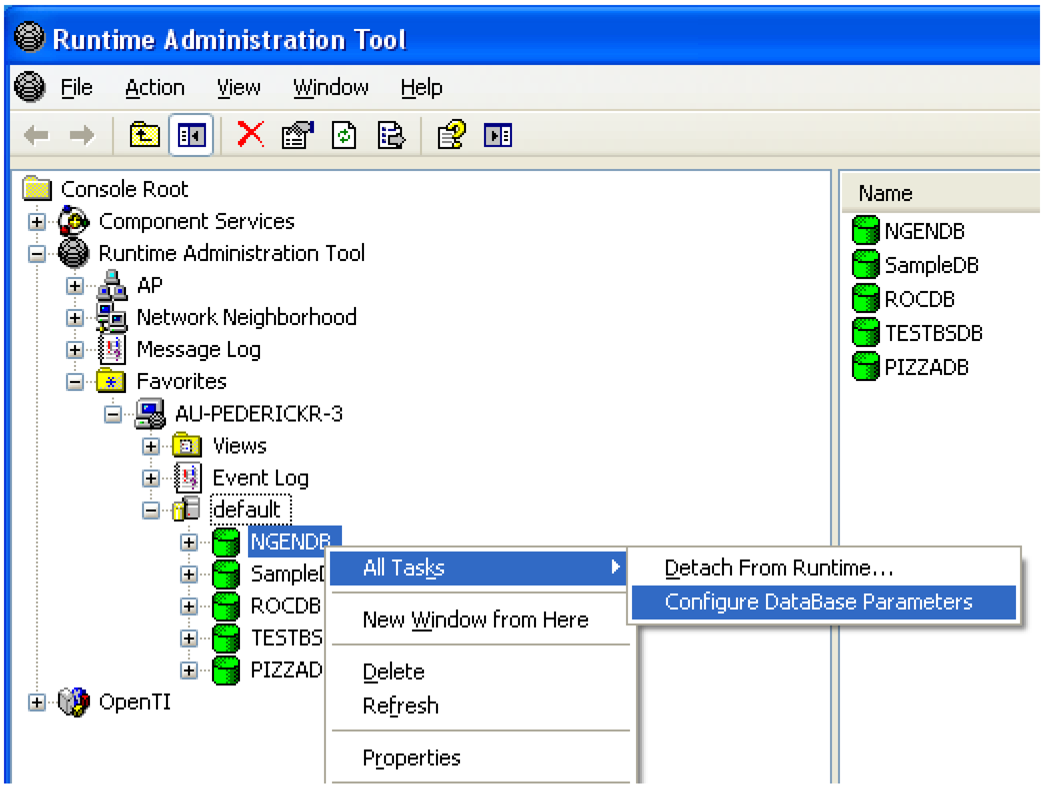
Note that this behaviour is actually similar to how EAE works with Oracle (on Windows and Unix). An Oracle feature called "Read Consistency" is used which states that the data at the start of a command will be the same for the length of the command regardless of whether it actually changes during the life of the command. Therefore, users migrating from EAE with Oracle will find that DataReader has similar behaviour.

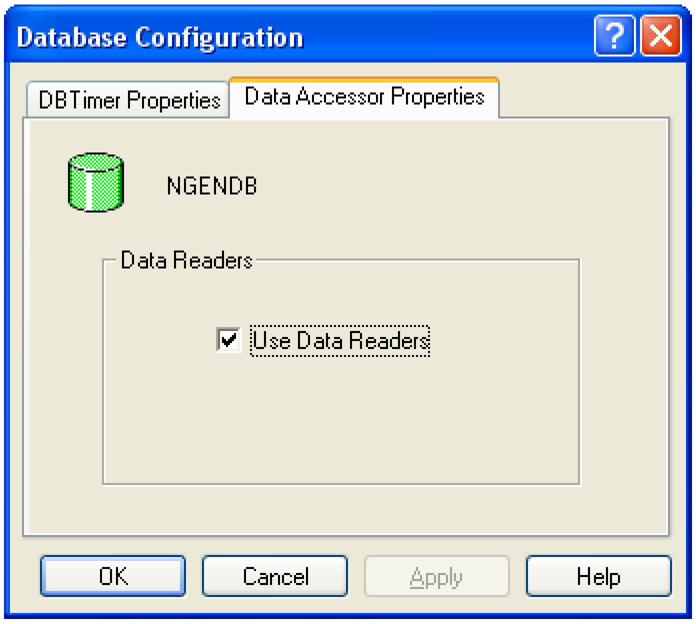
DataReader Runtime Settings

Apart from the **DataReader Capable** property that is used to build DataReader-capable code, it is possible to enable or disable DataReader at runtime for the system as a whole or for individual ispecs/reports.

A system-wide option is available in the Runtime AdminTool to enable/disable DataReader for the whole system. This option is set by default to enable DataReader.

Right-click the database node and select **All Tasks/Configure Database Parameters** and then select the **Data Accessor Properties** tab.





If this option is set (as it is by default) then all Determine commands that have been built as DataReader-capable will use DataReader at runtime. If this property is reset, then all Determine commands will use Cursors.

DBConfig.xml File

An XML configuration file called DBConfig.xml can be used to enable/disable DataReader for individual ispecs/reports. This XML file needs to be created within the AB Suite **Data\Public** folder (for example, C:\AB Suite 2.0\Data\Public). This file is not automatically created or generated, but will need to be manually created and the contents defined with a text editor. It will need to contain parent nodes to represent the database and the system within the database. The system node can contain attributes to enable/disable DataReader for the online system (all ispecs) and for all reports. Within the system node you can include child nodes to represent individual ispecs/reports. A DataReader setting at the individual ispec/report level will override the system-level setting.

An example of the **DBConfig.xml** file is shown below:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <database name="SAMPLEDB">
    <system name="Sample" DataReaderOnline="false" DataReaderReports="false">
      <ispecs>
        <ispec name="CUST" datareader="true" />
        <ispec name="SALE" datareader="false" />
        <ispec name="INGDS" datareader="true" />
        <ispec name="SINQ" datareader="false" />
      </ispecs>
      <reports>
        <report name="CONSOLIDAT" datareader="false" />
        <report name="CUSTLIST" datareader="true" />
        <report name="PRODSTATS" datareader="true" />
      </reports>
    </system>
  </database>
</Configuration>
```

In the `<system>` node the attribute `DataReaderOnline` is used to set a default DataReader setting for the online system i.e. all ispecs. The attribute `DataReaderReports` is used to set a default DataReader setting for all reports.

If you set the **System.log** logging level to "Information" (or "Debug") then you will get logged messages like the following showing that the DBConfig.xml file has been found, and that its contents are valid and have been successfully read. If you have created a DBConfig.xml file with DataReader settings, then it is important that you check the System.log after starting the system to verify that it is being read and interpreted correctly. Any XML syntax errors will be reported in the System.log and the system will continue running with default DataReader settings.

```
DataReader properties being read from C:\AB Suite 2.0\Data\public\DBConfig.xml
DataReaderOnline is set to False
DataReaderReports is set to True
Add to DataReaderMap: CUST True
Add to DataReaderMap: SALE False
```

Finer control of DataReader

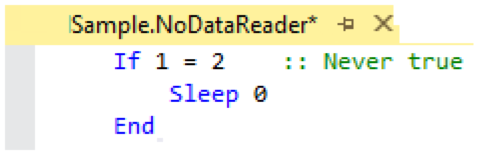
The previous section describes how you can control the use of DataReader at the whole ispec or report level. However, there might be situations where you have a report or ispec that does many Determine commands, most of which work best with DataReader, but there is one Determine command that does not work well with DataReader. Perhaps this Determine block of code includes logic to update the records being read and the result set changes during the course of the Determine, and therefore this command should not use DataReader.

The simple option is to turn DataReader off for the whole report using the DBConfig.xml file. However, it is possible to getting finer control of DataReader and effectively turn DataReader off for a single command. Here is how this can be done.

In the section **DataReader Limitations** above, we described how DataReader is internally disabled if the logic within the Determine block includes a **Sleep** command. Well, we can use this to good effect by including a 'dummy' **Sleep** command in the logic – that is a **Sleep** command included in a condition that is never true. The presence of the **Sleep** command in the logic will result in DataReader being disabled for that command, but because it is never executed it will not affect the behavior of the report.

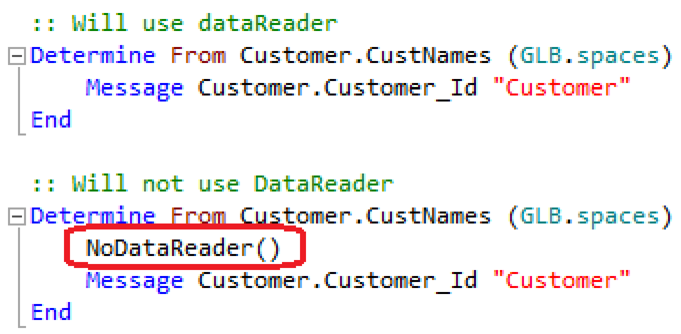
A recommended way of doing this is as follows:

Create a segment method like this.



```
If 1 = 2     :: Never true
    Sleep 0
End
```

Then call this method from inside any Determine command to effectively switch OFF DataReader. In the example below the first Determine will use DataReader and the second will not, because of the presence of a **Sleep** command. Note that the Sleep command does not need to be executed – it just needs to be present (anywhere within the block of code or within any method called from within the block – even nested deeply).



```
:: Will use dataReader
Determine From Customer.CustNames (GLB.spaces)
    Message Customer.Customer_Id "Customer"
End

:: Will not use DataReader
Determine From Customer.CustNames (GLB.spaces)
    NoDataReader()
    Message Customer.Customer_Id "Customer"
End
```

This will only affect the Determine commands that directly contain the Sleep. All other Determine commands in the same section of code (even nested commands) will be unaffected and will continue to use DataReader.

Examples of using DBConfig.xml

These examples assume that the **DataReader Capable** property has been defined and the system built to include DataReader-capable code for most Determine commands.

Example 1 - You want to use DataReader in all read loops across your entire system (ispecs and reports).

You would have no DBConfig.xml file, or no entries for the system.

DataReader will be enabled for all Determine commands across the system apart from those that contain a Sleep command.

Example 2 - You want to use DataReader in reports only.

The reason for this is that in general, reports will get more benefit from DataReader than ispecs. This is because reports often contain read loops that read through many records. Ispecs on the other hand are generally written to have quick response times, and typically read just a few records, so in general will not get as much benefit as reports.

You could include the following DBConfig.xml contents for this:

```
<Configuration>
  <database name="SAMPLEDB">
    <system name="Sample" DataReaderOnline="false" DataReaderReports="true">
    </system>
  </database>
</Configuration>
```

Example 3 - You want to use DataReader in all reports and also in a selected number of ispecs that read a lot of records and get a good benefit from DataReader.

You may have tested you system with DataReader on and off, and found that whereas most ispecs get no measurable benefit from DataReader, some selected ispecs that read a lot of records do get a significant benefit.

You could include the following DBConfig.xml contents for this:

```
<Configuration>
  <database name="SAMPLEDB">
    <system name="Sample" DataReaderOnline="false" DataReaderReports="true">
      <ispecs>
        <ispec name="ABISP" datareader="true" />
        <ispec name="IspecX" datareader="true" />
        <ispec name="CustomerInq" datareader="true" />
        <ispec name="SINQ" datareader="true" />
      </ispecs>
    </system>
  </database>
</Configuration>
```

Example 4 - You want to use DataReader in most reports but some reports show worse performance with DataReader.

You may have tested all reports with DataReader on and off, and found that whereas most reports get a significant benefit from DataReader, some reports actually show worse performance. You would want to disable DataReader for just these reports.

You could include the following DBConfig.xml contents for this:

```
<Configuration>
  <database name="SAMPLEDB">
    <system name="Sample" DataReaderOnline="false" DataReaderReports="true">
      <reports>
        <report name="CONSOLIDAT" datareader="false" />
        <report name="CUSTLIST" datareader="false" />
      </reports>
    </system>
  </database>
</Configuration>
```

Example 5 - You want to use DataReader in most reports but some reports show incorrect behaviour with DataReader.

The incorrect behaviour may be because the report contains a read loop in which records in the table being read are updated and added (see section “DataReader Behaviour” above). With DataReader, the subsequent read iterations do not see the updated records. Whereas with Cursors, the updated records are read correctly. In this case you will want to disable DataReader for these reports.

You could include the following DBConfig.xml contents for this:

```xml
<Configuration>
  <database name="SAMPLEDB">
    <system name="Sample" DataReaderOnline="false" DataReaderReports="true">
      <reports>
        <report name="CONSOLIDAT" datareader="false" />
        <report name="CUSTLIST" datareader="false" />
        <report name="VXREP5" datareader="false" />
        <report name="ZLISTREP" datareader="false" />
      </reports>
    </system>
  </database>
</Configuration>
```