NEXTGEN

VTEX Driver Migration

Kit Ling

Rev 0.1

Revision History

Revision	Date	Description	Author
0.1	May 29, 2009	Initial revision	Kit Ling

Table of Content

1 Objectives

The objectives of this document are to show:

1. The difference between the VTEX driver in Windows and Linux
2. How to migrate programs written in Windows to Linux and vice-versa

2 Difference & Migration

In this section, it shows the main differences between the VTEX COM driver (Windows) and the VTEX LIB driver (Linux). It also shows how to migrate programs from Windows to Linux.

2.1 DLL Import

C# Windows: Use the specific driver interop (make sure that you have already added the specific driver DLL in the reference section in the project prior to this)

```
Using VTI.VTEXDmm.Interop;
```

C++ Windows: Import the COM driver DLL files

```
#import “IviDriverTypeLib.dll” no_namespace
#import “IviDmmTypeLib.dll” no_namespace
#import “VTEXDmm.dll” no_namespace
```

Linux: Include the driver header file (make sure that you have already linked the specific driver shared objects in the Makefile prior to this)

```
#include “libDmm.h”
```

2.2 Driver pointer instantiation

C# Windows: Create and instantiate a new object of the specific driver data type

```
VTEXDmm dmm = new VTEXDmm();
```

C++ Windows: Start the COM layer and then instantiate a specific driver pointer

```
::CoInitialize(NULL); //Start the COM layer
IVTEXDmmPtr dmm(__uuidof(VTEXDmm)); //Instantiate a Dmm
driver pointer
```

Linux: Create a specific driver pointer and then instantiate the driver pointer

```
LibDmm *pDmm; //Create a Dmm driver pointer
pDmm = LibDmm::Create(); //Instantiate the created Dmm
driver pointer
```

2.3 Driver pointer deletion

C# Windows: Not required

C++ Windows: Set the driver pointer to NULL and then uninitialize the COM layer.

```
pDmm = NULL; //Set the driver pointer to NULL
::CoUninitialize(); //Uninitialize the COM layer
```

Linux: Delete the driver pointer and then set it to NULL

```
delete(pDmm); //Delete the driver pointer
pDmm = NULL; //Set the driver pointer to NULL
```

2.4 Data type

The COM driver (Windows) and the LIB driver (Linux) use different data types and values. Table 2.4.1 shows the summary of all the different data types and values.

C# Windows		C++ Windows		Linux	
Data Types	**Values**	**Data Type**	**Values**	**Data Type**	**Values**
bool	true	VARIANT_BOOL	VARIANT_TRUE	bool	true
bool	false	VARIANT_BOOL	VARIANT_FALSE	bool	false
string	“sample string”	_bstr_t	“sample string”	std::string	“sample string”
int	1	HRESULT	S_OK	VTEXResult	IVI_SUCCESS
COMException	e.ErrorCode	_com_error	e.Error()	VTEXException	e.errorCode
COMException	e.Message	_com_error	e.ErrorMessage()	VTEXException	e.errorMessage
COMException	---	_com_error	e.Description()	VTEXException	---
COMException	---	_com_error	---	VTEXException	e.timeStamp

Table 2.4.1. Summary of the data types and values in COM and LIB driver

2.5 Array, SAFEARRAY and std::vector

For some functions, the COM driver uses Array (in C#) and SAFEARRAY (in C++) to store data while the LIB driver uses std::vector. For examples, VTEXDmm driver FetchMultiPoint function uses Array (in C#) and SAFEARRAY (in C++) in the COM driver but std::vector in the LIB driver as the second parameter. For more information about similar functions, please read the user manual.

C# Windows:

```
FetchMultiPoint(long MaxTimeMilliseconds, ref double[]
ReadingArray);
```

C++ Windows:

```
FetchMultiPoint(long MaxTimeMilliseconds, SAFEARRAY**
ReadingArray)
```

Linux:

```
FetchMultiPoint(long MaxTimeMilliseconds, vector<double>
&ReadingArray)
```