IVI-COM Instrument Driver Programming Guide (IVI Config Utility Edition)

Dec 2003 Revision 1.0

1- Overview

One of features of IVI instrument drivers is the interchangability function. By using the interchangability function, applications do not have to be recompiled or re-linked to continue to work even if an instrument has been swapped with other model.

To utilise the interchangability function, there must be IVI-COM instrument drivers provided for both pre-swapped and post-swapped instruments. Plus, they both must be IVI-COM class-compliant drivers that comply with the same instrument class. There are no interchangability capabilities between different instrument classes. Also, an application that utilises the interchangability function must indirectly access the instrument drivers through the class interfaces, not through the driver-supplied specific interfaces.

When using the interchangability function, detailed setups regarding the instrument (and its driver) being actually used have to be placed in an external storage outside the application. This external storage is what called the IVI Configuration Store. In the IVI Configuration Store, information about installed instrument drivers and virtualised instruments associated with them is placed.

In this document, we explain how to use the **Kikusui IVI Config Utility**, which configures virtual instrument settings.

1-1 Virtual Instrument

What you have to do before creating an application that utilises interchangability function is create a virtual instrument. In your application, you should not write codes that are specific to a particular IVI-COM instrument driver (e.g. creating an object directly with Kikusui4800 type, or writing a VISA resource name such as "GPIB0::3::INSTR"). Writing these kinds of codes spoils interchangability.

Instead, IVI-COM specifications provide interchangability mechanisms by placing the IVI Configuration Store outside instrument drivers and applications. An application indirectly select an instrument driver according to the contents of the IVI Configuration Store, and accesses the indirectly loaded instrument driver through the class interfaces that are independent of particular instrument.

IVI Configuration Store is normally the /Program Files/IVI/Data/IviConfigurationStore.XML file and accessed through the IVI Configuration Server DLL. Software that utilise this DLL are mainly IVI-COM instrument drivers and configuration tools provided by instrument and/or instrument driver vendors. Applications normally do not use it. Kikusui provides a configuration tool - **Kikusui IVI Config Utility**. By using this, you can configure virtual instrument settings.

Notes:

A virtual instrument is identified by a Logical Name.

This guidebook assumes that you use the IVI-COM Kikusui4800 Instrument Drivers (for Kikusui PIA4800 series DC Power Supply Controller). You can also use other versions of IVI-COM instrument drivers in the same manner.

2- Kikusui IVI Config Utility

2-1 Launching The Utility

From the **[Start] button➔Programs➔IVI➔Kikusui IVI Config Utility** menu, you can launch the **Kikusui IVI Config Utility**. The start-up screen looks like the following picture. The left side is a tree-view, which displays two hierarchies. The upper side is **Software Modules** on which all the installed IVI-COM instrument drivers are displayed. Drivers from other than Kikusui, such as from Agilent Technologies or Tektronix, are also displayed. The lower side is **Logical Names** on which all the available virtual instruments are displayed. In the following picture, a virtual instrument having a logical name **Kikusui4800** is configured.

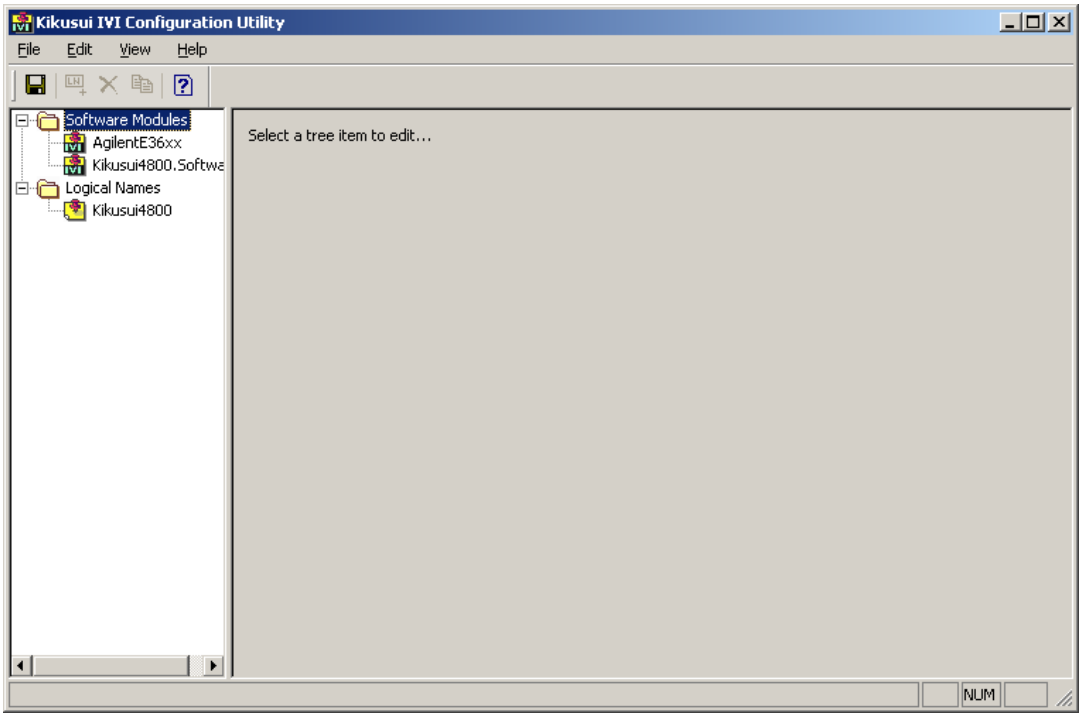

Figure 2-1 Kikusui IVI Config Utility (Main Screen)

The logical name Kikusui4800 represents the default virtual instrument, which was created when the Kikusui4800 IVI-COM driver was installed. Although you can customise it, this document here provides an example for how to create new one.

Notes:

By setting up an IVI-COM instrument driver, the default virtual instrument (logical name) may be created. This is normally provided as an example for creating a virtual instrument. May be good to utilise it by changing the logical name.

Depending on IVI-COM driver versions or vendors, such default virtual instrument may not be provided.

2-2 Adding A Logical Name

Right-click the part where the label **Logical Names** is shown on the tree to make the context menu appear, then select **Add Logical Name...**. The **Add Logical Name** dialog appears.

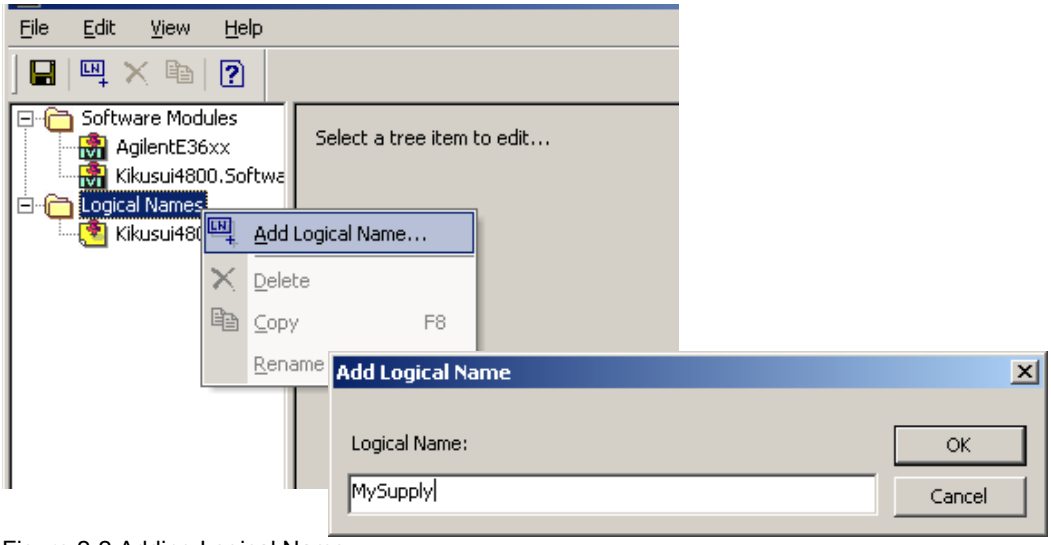

Figure 2-2 Adding Logical Name

Here you give a virtual instrument name that you want to create. Letters that can be used for a logical name are alphanumeric (A..Z, a..z, 0..9), underscore (_), and exclamation (!) only. Logical names that already exit cannot be used. Logical names are case-sensitive.

As an example, give the name "`MySupply`". Then this name is added to **Logical Names** and displayed. After creating a virtual instrument giving a logical name, the next steps are configuration work on the tab pages - **Logical Name**, **Driver Session**, **Hardware Asset**, and **Virtual Names**.

2-3 Configuration On Tab Pages

Logical Name Tab

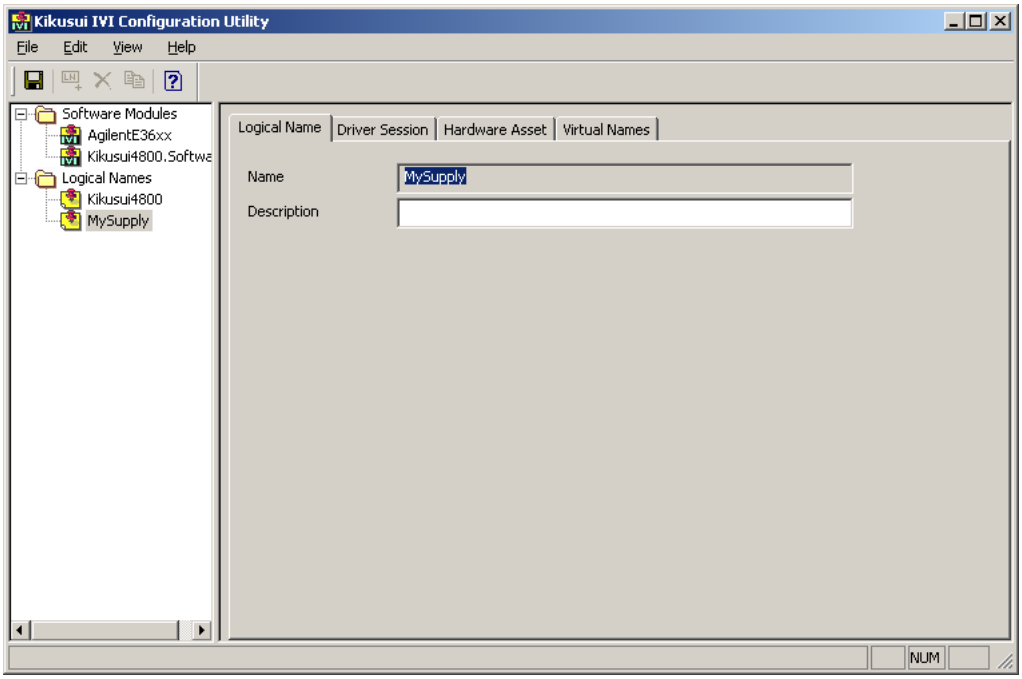

Figure 2-3 Logical Name Tab

What you can configure on this page is **Description** only. It has no special meaning for functionality therefore skip it here. You can leave it empty.

Driver Session Tab

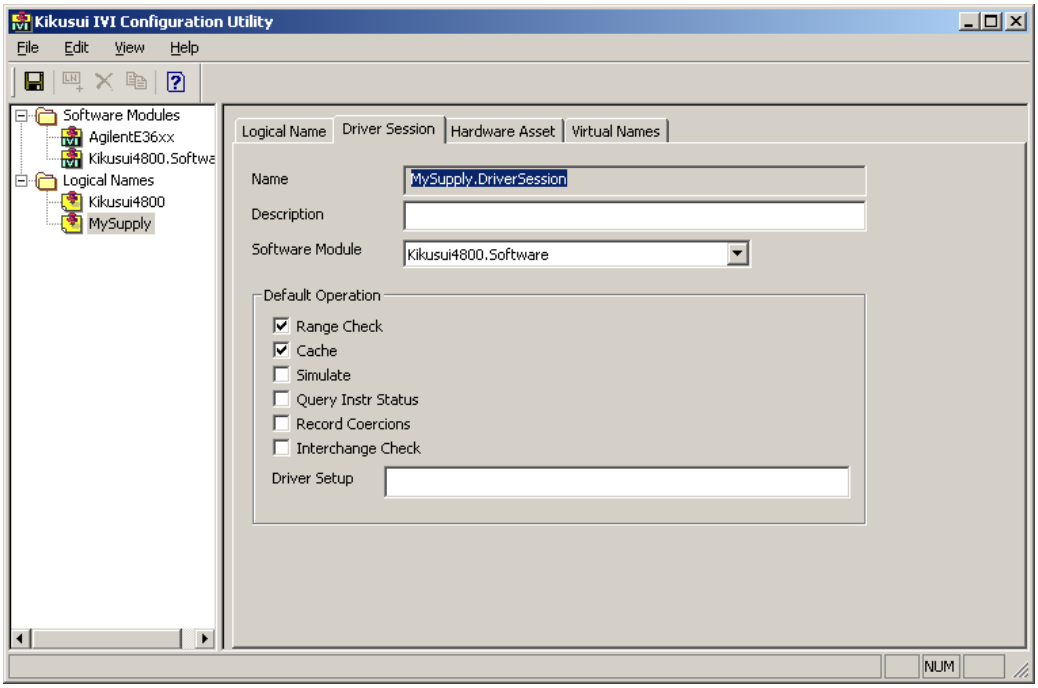

Figure 2-4 Driver Session Tab

Description has no special meaning for functionality so you can leave it empty. **Software Module** selects a software module enumerated by the combobox. This is a very important setting that decides what instrument driver is used for hosting the virtual instrument. This example uses Kikusui4800.Software as the hosting driver, therefore select it. You must select at least one item.

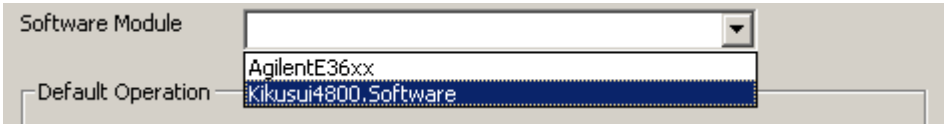

Figure 2-5 Software Module Selection

At **Default Operation**, you configure ON/OFF settings for **Range Check**, **Cache**, **Simulate**, **Query Instr Status**, **Record Coercions**, and **Interchange Check**, plus the **Driver Setting** string. The contents that you configure here are used as the default settings when the application invokes the `Initialize` method.

Figure 2-6 Default Operation Settings

If **Range Check** is enabled, the driver validates the given input value for property's put_ access and the given input parameters for method calls. This feature works as a pre-

verification work before sending out-of-range values to the instrument. This is useful when the application is under debugging.

If **Cache** is enabled, the driver avoids unnecessary I/Os. For example, if a setting value is already sent to the instrument, it is redundant to send the same value again. By avoiding these wasting I/Os, application's performance increases.

If **Simulate** is enabled, the driver does not perform I/O to the instrument, and returns simulated values for output parameters. This is useful when the actual instrument set is not ready for application development.

If **Query Instr Status** is enabled, the driver queries the instrument status at the end of each method call or property access that performs I/O to the instrument. If an instrument error is reported, you can use the `ErrorQuery` method to retrieve error messages from the instrument. This is useful when the application is under debugging, but it is better to be disabled after the development is completed.

If **Record Coercions** is enabled, the driver keeps a list of the value coercions regarding LONG and DOUBLE values. If the driver does not support coercion recording, this setting is ignored.

If **Interchange Check** is enabled, the driver maintains a record of interchangability warnings. If the driver does not support interchangability checking, this setting is ignored.

Hardware Asset Tab

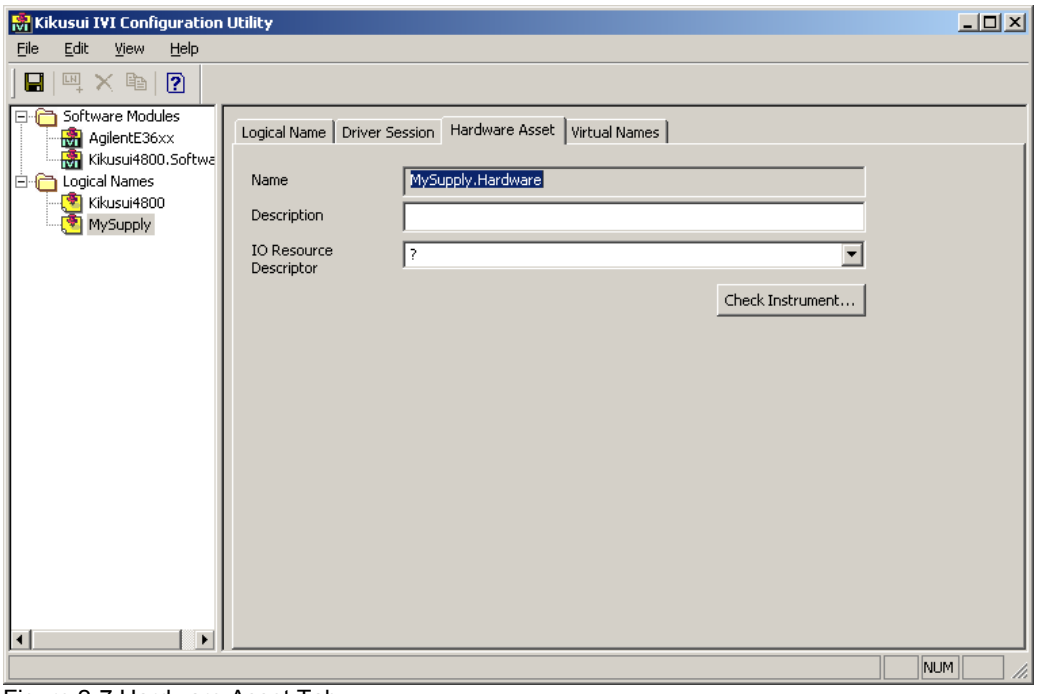

Figure 2-7 Hardware Asset Tab

Description has no special meaning for functionality so you can leave it empty. The combobox of **IO Resource Descriptor** enumerates VISA I/O resources that are currently available. Select an appropriate item. Instruments that are not connected or connected through the TCP/IP are not shown. In this case, input a valid VISA resource directly. When the application invokes `Initialize` method passing the logical name as a parameter, the driver performs instrument I/Os though the VISA resource configured here.

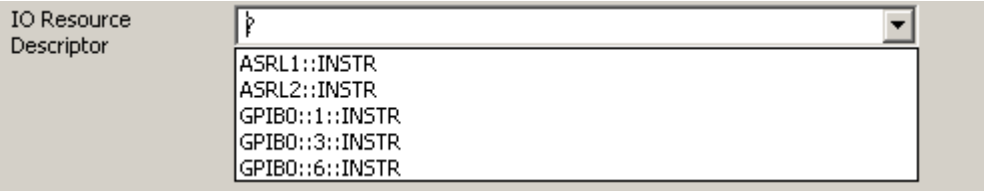

Figure 2-8 IO Resource Descriptor Selection

If you click the **Check Instrument** button, you can perform a simple I/O test using the VISA COM library.

Virtual Name Tab

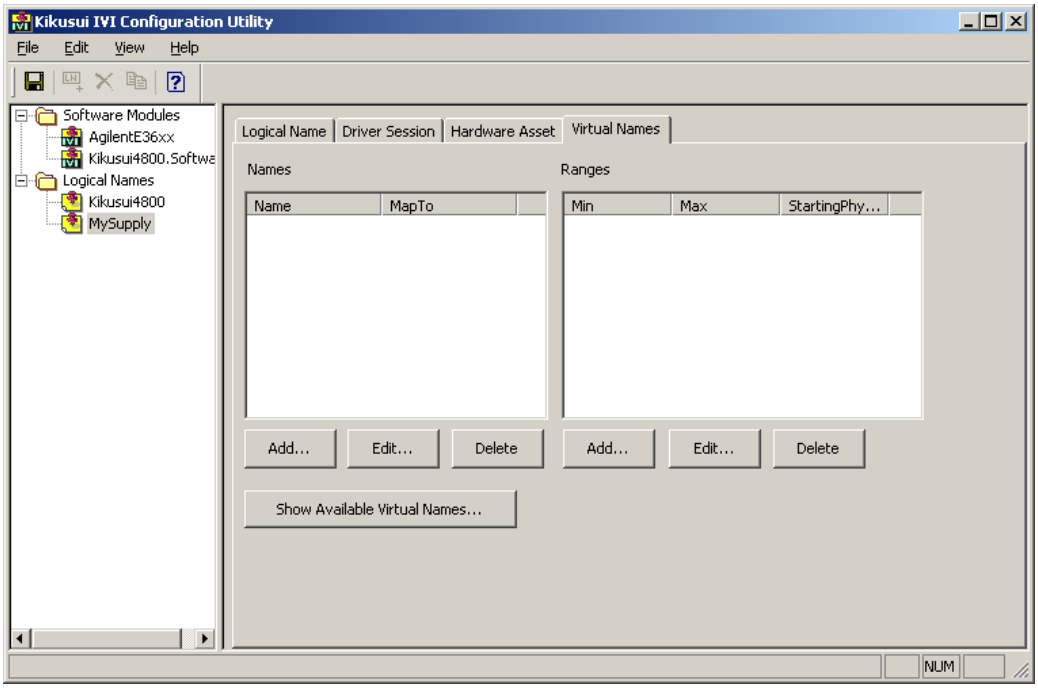

Figure 2-9 Virtual Names Tab

On this page, you configure mappings between virtual names and physical names. The settings here are used for identifications of objects provided by **Repeated Capabilities** feature.

Repeated Capabilities are a concept that treats the same or identical multiple objects as if they are in an array or a container. For example, an instrument driver that complies with the IviDCPwr class is designed as a multi-track DC power supply driver assuming that the instrument has multiple output channels. Another example is, an instrument driver that complies with the IviScope class is designed as an oscilloscope driver assuming that the instrument has multiple trace channels. Similarly it is recommended to use Repeated Capabilities in the case that there are identical multiple objects, according to the IVI specifications.

Although the Kikusui4800 IVI-COM driver that complies with IviDCPwr class identifies output objects by name having a naming convention like "N5!C1", such names are driver-specific. This kind of name is called Physical Name. However an application that utilises interchangability function should not use physical names, which have dependency on particular instrument driver. To avoid this, you need create virtual names that are mapped to driver-specific physical names.

By clicking the **Add** button at the **Names** side, the **Virtual Name** dialog appears.

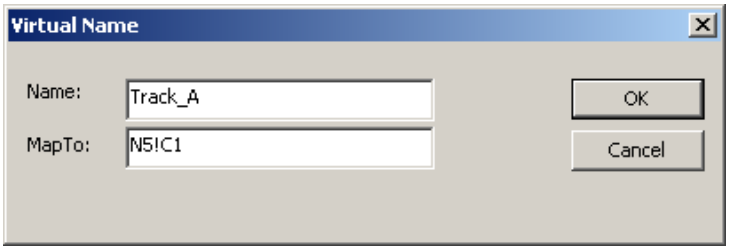

Figure 2-10 Virtual Name Dialog

As an example, type `Track_A` for **Name**, and type `N5!C1` for **MapTo**, then click the **OK** button. Now a virtual name has been created.

By this configuration, when the application attempts to reference the "`Track_A`" object through the repeated capabilities, then the instrument driver understands the mapped name "`N5!C1`" shall be referenced. Thereby your application can reference the object by "Track_A" instead of by "N5!C1", which actually existed as a physical name in the driver.

In the same manner, create more mappings as you need. You cannot use the same name for **Name** more than once. Every virtual name must be unique. However, you can map multiple different virtual names to the same physical name. It means you can create aliases with multiple virtual names mapped to the same physical name.

You do not have to create virtual names corresponding to all the physical names for each. For example, "N30!C4" is a valid physical name in the Kikusui4800 IVI-COM driver, but you do not have to create virtual names that are mapped to it unless your application references its object.

To show all the available virtual names and corresponding physical names, click the **Show Available Virtual Names** button.

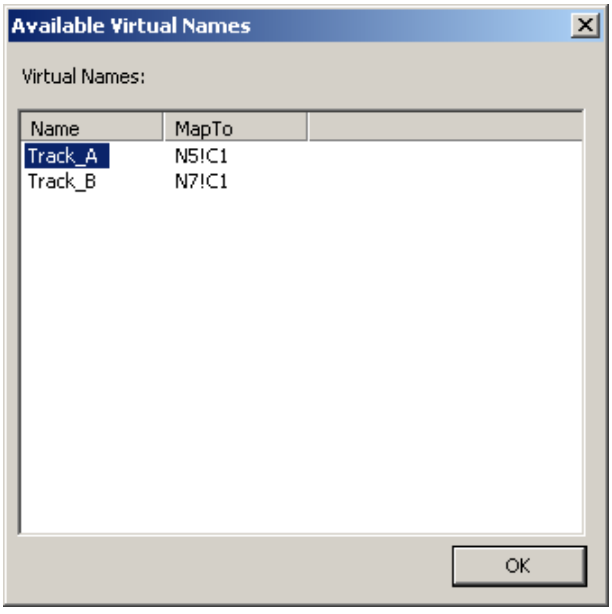

Figure 2-11 Show Available Virtual Names Dialog

Physical names that can be specified as the mapped targets are completely different depending on the software module (instrument driver software) that was selected by the **Software Module** combobox on the **Driver Session** tab. To obtain the information about this, you need refer to the online help or the Readme document of each instrument driver. You can also retrieve the information by using the sample codes that are introduced in the guidebooks for each programming language.

2-4 Saving Configuration

After completing the virtual instrument configuration, save the settings by **File | Save** menu.

IVI-COM Instrument Driver Programming Guide