

IBM Dynamic Application Virtualization
(IBM DAV)



IBM Dynamic Application Virtualization User Guide alphaWorks update March 2009

IBM Dynamic Application Virtualization
(IBM DAV)



IBM Dynamic Application Virtualization User Guide alphaWorks update March 2009

Contents

Figures v

Tables vii

Preface ix

About this document xi

Who should use this document xi

Typographic conventions xi

Related information xi

Chapter 1. Introducing IBM Dynamic Application Virtualization 1

Overview 1

What is IBM Dynamic Application Virtualization? 1

Chapter 2. IBM DAV Platform Requirements 3

Operating Systems 3

Software Requirements 3

Chapter 3. IBM DAV installation instructions 5

IBM DAV Installation Components 6

Client-side installation for Windows 6

Client-side installation for Linux 6

Client-side developer installation for Windows 6

Client-side developer installation for Linux 7

Server-side installation for Linux 7

Chapter 4. IBM DAV Virtualizer Configuration 9

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005 Express Edition 9

 Installing Microsoft Visual C++2005 Express Edition 9

 Installing MS Platform SDK (R2) 10

 Installing the Microsoft Visual C++ 2005 Redistributable Package 10

 Configuring IBM DAV Virtualizer to interact with MS Visual Studio 10

 Linking Microsoft Visual C++ and the Microsoft Platform SDK 10

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005 11

 Configuring IBM DAV Virtualizer to interact with MS Visual Studio 11

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2003 12

 Configuring IBM DAV Virtualizer to interact with MS Visual Studio 2003 12

Configuring IBM DAV Virtualizer for Linux using GCC 4.1.2 12

Set the IBM_DAV_PATH and LD_LIBRARY_PATH 12

Chapter 5. IBM DAV Semantics 15

IBM DAV Comment Format 15

Tag Types 15

 Library tag 15

 Function tag 16

 Function Tag: @param 16

 Function tag @dimensions 18

 Function tag @return 18

 Struct tag 18

 Void* 19

 Strings 20

IBM DAV Dimension definitions 20

Chapter 6. IBM DAV Virtualizer Usage 23

Generating DAV Libraries 23

Sample Libraries 24

 BLAS 24

 LAPACK 25

 FFT 26

 Monte Carlo (MC) 26

Library Semantics 28

Chapter 7. Semantic Verification 29

Chapter 8. Server Side Deployment 33

Automatic Deployment 33

Chapter 9. Running the Server-side IBM DAV Service 35

Server-side IBM_DAV.conf 35

Start the Service Broker 35

Launch the Service 35

Verify the service is running using 'dav status' 36

Stopping the IBM DAV Service 36

Running a Service - Dedicated Mode 36

Setting an alternative location for the shared object 36

Chapter 10. IBM DAV Virtualizing the Client Application 39

Virtualizing a C++ Client Application on Linux 39

Virtualizing a C++ Client Application on Windows 39

 Startup 40

 Set additional lib path 40

 Set additional Include path 40

 Add the IBM DAV generated library to the dependencies 41

 Build the Client Application 41

Virtualizing an Excel Client Application 41

Virtualizing a Java Client Application 42

Chapter 11. Running the client application	43
Update the server settings in the client-side IBM_DAV.conf	43
Launch the Application	44
Running an Excel application	44
Running a Java application	44
Chapter 12. XLL Add-ins	47
Enabling XLL Add-in Functionality	47
XLL Use	47
Using XLL Add-in Functionality - Example.	48
XLL Limitations	49
XLL Assumptions	50
Appendix A. Sample Header File.	51
Appendix B. Semantics - Sample Header File	53

Appendix C. Server Side IBM_DAV.conf settings	57
Appendix D. Client Side IBM_DAV.conf settings	59
Appendix E. IBM DAV Executable Options	61
Appendix F. makefile for IBM DAV Virtualizing a Linux application	63
Appendix G. Troubleshooting Errors	65
Index	67

Figures

Tables

1. Downloadable files available for IBM DAV	5	4. How the Original Interface is accessed using	
2. Supported C operators	20	DAV	27
3. General Usage for davGen.sh/ davGen.bat	24	5. General Usage deployer.sh/ deployer.bat	34

Preface

This document is intended for users who wish to install, configure and use the IBM® Dynamic Application Virtualization (IBM DAV) product.

About this document

This document describes installation, configuration and operation for IBM Dynamic Application Virtualization.

Who should use this document

This document is designed for:

- System administrators
- Developers - to install, configure and operate IBM DAV

Typographic conventions

This document uses the following typographic conventions:

- Commands appear in bold font when they appear in text.
 - Example: **logon**
- File and directory names are italicized when they appear in text.
 - Example: */etc/passwd*
- Variables for user-supplied information on commands are indicated by italics.
 - Example: **telnet** *app_host*
- Literals in commands appear in monospace font.
 - Example: `rpm -i --force abc.rpm`
- Angle brackets (< >) indicate that you must make a substitution.
 - Example: <username> indicates that you must supply your user name.
- Square brackets ([]) indicate that the enclosed values are optional.
- Curly brackets ({ }) indicate that you must select one of the enclosed values.
- Multiple options for a value are separated by a vertical bar (|).
- Default values are underlined.

Related information

For the further information about IBM DAV, please visit the IBM Dynamic Application Virtualization alphaWorks website at <http://www.alphaworks.ibm.com/tech/dav/>

Chapter 1. Introducing IBM Dynamic Application Virtualization

Overview

IBM Dynamic Application Virtualization is a technology that enables computationally-intensive applications to take advantage of accelerated libraries on remote, backend systems including Cell Broadband Engine (Cell/B.E), reducing time to deployment and disruption to business.

What is IBM Dynamic Application Virtualization?

IBM Dynamic Application Virtualization provides the ability to offload function calls in a computationally intensive application to remote, high-performance computation nodes, thereby significantly reducing time to deployment, with minimal code changes and disruption to business.

In financial services, IBM Dynamic Application Virtualization can help developers to quickly deploy new applications that take advantage of accelerated math libraries running on new or remote hardware systems. In particular, where original applications were not written to offload calculations to remote hardware, IBM Dynamic Application Virtualization can be used to prepare the application for invoking remote libraries, thereby avoiding a rewriting or customization of the original code base.

Specifically, this technology can cut down overall time to deployment for business-critical applications that require ultrahigh speed and low latency; such applications include options pricing (for example, Monte Carlo simulations) and FIX message parsing. In the past, such applications may have required significant application development time, prolonging the potential for a return on new investment, sometimes making the project cost-prohibitive. By using IBM Dynamic Application Virtualization to call the remotely deployed application functions residing on high-performance computation nodes, combined with the low-latency infrastructure provided by IBM Dynamic Application Virtualization, application calculation times can also be reduced.

The technology virtualizes application libraries by examining the library's exported functions and capturing additional semantics. Library functions, when invoked from an application, appear to be executed locally, insulating developers from the specifics of the computation nodes and reducing the effort involved in calling remote functions.

IBM Dynamic Application Virtualization supports applications written in C/C++, Java, and Excel spreadsheets (Visual Basic Application (VBA)), and it harnesses the power of the IBM BladeCenter QS21, QS22 with Cell/B.E. The technology promotes heterogeneous adaptation; that is, it allows client applications running on one architecture (such as Intel/Windows XP) to execute functions hosted on a different architecture (such as Cell/B.E).

Chapter 2. IBM DAV Platform Requirements

Operating Systems

Client

For the client one of the following is required

- Microsoft Windows XP SP2
- Red Hat Enterprise Linux Client 5.1
- SUSE Linux Enterprise Server 10 for x86_64

Server

For the server one of the following

- Red Hat Enterprise Linux 5.1,5.2 on x86 nodes
- Red Hat Enterprise Linux 5.1,5.2 on IBM BladeCenter QS21
- Red Hat Enterprise Linux 5.2 on IBM BladeCenter QS22.

Software Requirements

Windows clients

For Windows clients, one of the following must be installed

- Microsoft Visual Studio Professional 2005 (recommended)
- Microsoft Visual Studio Professional 2003
- Microsoft Visual Studio Express 2005 and Microsoft Platform SDK

Linux clients

For the Linux clients

- libXp
- gtk+
- compat-libstdc++-33
- unixODBC
- gcc-c++.i386
- gcc4.1.2-14.e15
- libXp.so.6 is needed by dav-virtualizer-0.1-1.i386
- ibgdk-1.2.so.0 is needed by dav-virtualizer-0.1-1.i386
- libgmodule-1.2.so.0 is needed by dav-virtualizer-0.1-1.i386
- libgtk-1.2.so.0 is needed by dav-virtualizer-0.1-1.i386
- libodbc.so.1 is needed by dav-virtualizer-0.1-1.i386
- libstdc++.so.5 is needed by dav-virtualizer-0.1-1.i386
- libstdc++.so.5(CXXABI_1.2) is needed by dav-virtualizer-0.1-1.i386
- libstdc++.so.5(GLIBCXX_3.2) is needed by dav-virtualizer-0.1-1.i386

Chapter 3. IBM DAV installation instructions

Installation files for IBM DAV are on the IBM DAV alphaWorks website

- <http://www.alphaWorks.ibm.com/tech/dav/>

The following files are available

Table 1. Downloadable files available for IBM DAV

Filename	Description
<i>dav-server.i386.rpm</i>	IBM DAV Server side software, installed on the remote x86 node where the application library will be deployed for computation of client requests.
<i>dav-server.s390.rpm</i>	IBM DAV Service Broker software, installed on zLinux.
<i>dav-server.ppc64.rpm</i>	IBM DAV Server side software, installed on the remote IBM BladeCenter QS2x node where the application library will be deployed for computation of client requests.
<i>dav-client.i386.rpm</i>	IBM DAV Client side software for Linux, installed on the client where the requesting application resides.
<i>dav-client.s390.rpm</i>	IBM DAV Client(31 bit) side software for zLinux, installed on the client where the requesting application resides.
<i>dav-client.x86_64.rpm</i>	IBM DAV Client side software for SLES Linux x86_64. Installed on the client where the requesting application resides.
<i>dav-virtualizer.i386.rpm</i>	IBM DAV Virtualization software for Linux, installed on the client where the application developer will use the Virtualization component to generate client and server side code to prepare the application for function off-loading. The client software is also a pre-requisite to running IBM DAV Virtualizer.
<i>dav-virtualizer.s390.rpm</i>	IBM DAV Virtualizer software for zLinux, installed on the client where the application developer will use the Virtualizer component to generate client and server side code to prepare the application for function offloading. The client software is also a prerequisite to running the IBM DAV Virtualizer.
<i>davClientInstall.exe</i>	IBM DAV Client side software for Windows, installed on the client where the requesting application resides.
<i>davVirtualizerInstall.exe</i>	IBM DAV Virtualizer software for Windows, installed on the client where the application developer will use the Virtualizer component to generate client and server side code to prepare the application for function off-loading. The client software is also a pre-requisite to running IBM DAV Virtualizer.

Table 1. Downloadable files available for IBM DAV (continued)

Filename	Description
<i>IBM_DAV_Demo.zip</i>	A short movie demonstrating how to run the BLAS function DGEMM from an Excel spreadsheet using IBM DAV to remotely execute the function on a Cell/B.E. BladeCenter.

IBM DAV Installation Components

The IBM DAV Virtualizer install includes:

- IBM DAV DavGen
- IBM DAV Deployer
- Eclipse CDT 4.0.0
- IBM JRE 5.0 SR4
- Ant 1.7.0
- JSch 0.1.31

The IBM DAV Client install includes:

- IBM DAV Client
- Sample Application – client-side library source code
- Sample Application – client-side library binaries

The IBM DAV Server rpm includes:

- IBM DAV Server
- Sample Application – server-side library source code
- Sample Application – server-side library binaries

Client-side installation for Windows

1. Download and install *DavClientInstall.exe* on the client machine where the target application resides.
2. Follow the instructions in the installer. By default, the client will be installed to
 - *C:\Program Files\IBM\IBM DAV*

Client-side installation for Linux

1. Download and install *dav-client.i386.rpm* on the client machine where the target application resides.
2. Log in with root access and run the following command:
 - `rpm -ivh dav-client.i386.rpm`
3. By default, the client will be installed to
 - */usr/ibmdav/client*

Client-side developer installation for Windows

1. Download and install *DavVirtualizerInstall.exe* on a client machine (This component will be used to generate client- and server-side code in order to prepare the application for function off-loading).

Note: The client can be the same machine on which the target application resides or a different machine, but the client software is required for running the Virtualizer component of IBM Dynamic Application Virtualization.

2. Follow the instructions in the installer. The software will be installed to .
 - *C:\Program Files\IBM\IBM DAV\Virtualizer*

Client-side developer installation for Linux

1. Download and install *dav-virtualizer.i386.rpm* on a client machine. (This component will be used to generate client- and server-side code in order to prepare the application for function off-loading.)

Note: The client can be the same machine on which the target application resides or a different machine, but the client software is required for running the Virtualizer component of IBM Dynamic Application Virtualization.

2. Log in with root access and run the following command:
 - *rpm -ivh dav-virtualizer.1386.rpm*
3. By default, this software will be installed to
 - */usr/ibmdav/virtualizer*

Server-side installation for Linux

1. Download *dav-server.i386.rpm* for x86 server machines or *dav-server.ppc64.rpm* for IBM BladeCenter QS21, QS22.
2. Log in with root access and run the following command:
 - *rpm -ivh dav-server.i386.rpm* or *rpm -ivh dav-server.ppc64.rpm*

The server software will be installed to */usr/ibmdav/server*.

Chapter 4. IBM DAV Virtualizer Configuration

About this task

On **Windows® XP**, IBM DAV Virtualizer can be configured to run with any of the following:

- Microsoft® Visual C++ 2005 (8) Express Edition, see “Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005 Express Edition”
- Microsoft Visual C++ 2005 (8), see “Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005” on page 11
- Microsoft Visual Studio 2003 (7), see “Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2003” on page 12

Note: For the purposes of the document, screenshots refer to Microsoft Visual C++ 2005 Express Edition.

On **Linux®**, IBM DAV Virtualizer can be configured to run with:

- GCC 4.1.2 see “Configuring IBM DAV Virtualizer for Linux using GCC 4.1.2” on page 12

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005 Express Edition

Before you begin

Assumptions:

IBM DAV Client has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

IBM DAV Virtualizer has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

Note: See “IBM DAV Installation Components” on page 6 for components installed by the above

Installing Microsoft Visual C++2005 Express Edition

1. Download Microsoft Visual C++ 2005 Express Edition from the Microsoft Visual Studio 2005 Express Editions website at:
<http://www.microsoft.com/express/2005/download/default.aspx>
2. Run *vcsetup.exe* and install to a user preferred area
For Example: *C:\Program Files\Microsoft Visual Studio 8*
3. Launch **Microsoft Visual C++ 2005 Express Edition** to open its start page.

Note: For Microsoft Visual C++ 2005 Express Edition, the MS Platform SDK (R2) and the MS Visual C++ 2005 Redistributable Package need to be installed. Details are given in the following sections.

Installing MS Platform SDK (R2)

1. Go to the Microsoft Windows Server 2003 R2 Platform SDK Web Install website at:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=0BAF2B35-C656-4969-ACE8-E4C0C0716ADB&displaylang=en>
2. On the Microsoft Windows Server 2003 R2 Platform SDK Web Install page - click **Continue** and perform Microsoft Validation.
3. On the updated **Windows Server 2003 R2 Platform SDK Web Install** page - click **Download Files Below** – this brings you to the **Files in this Download** section of the page.
4. Download the *PSDK_x86.exe* file
5. Run *PSDK_x86.exe* then follow the installation wizard to install platform SDK

Installing the Microsoft Visual C++ 2005 Redistributable Package

1. Download **Microsoft Visual C++ 2005 Redistributable Package** (*vc redistrib_x86.exe*) from the Microsoft Visual C++ 2005 Redistributable Package (x86) webpage at:
<http://www.microsoft.com/downloads/details.aspx?familyid=32bc1bee-a3f9-4c13-9c99-220b62a191ee&displaylang=en>
2. Install the Microsoft Visual C++ 2005 Redistributable Package.

Configuring IBM DAV Virtualizer to interact with MS Visual Studio

About this task

If Microsoft Visual Studio C++ is installed to a directory other than *C:\Program Files\Microsoft Visual Studio 8* the following steps need to be performed:

1. Navigate to the IBM DAV Virtualizer install location (to the user preferred area specified in the installation of IBM DAV Virtualizer) and open the *preference.ini* file.

For Example:

C:\Program Files\IBM\IBM DAV\Virtualizer\davGen

2. Edit the entry
virtualizer.compiler.rootfolder.msvc = C:\Program Files\Microsoft Visual Studio 8
to point to the install location for C++.

Linking Microsoft Visual C ++ and the Microsoft Platform SDK

This task describes the steps required to ensure that Microsoft Visual C++ 2005 Express Edition recognizes the Microsoft Platform SDK.

Before you begin

Assumption It is assumed that Microsoft Visual C++ 2005 Express Edition is installed to *C:\Program Files\Microsoft Visual Studio 8*. Please replace this default area with the user preferred area if it was set differently in “Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005 Express Edition” on page 9

About this task

The file *vsvars32.bat* has to be modified so that it will match the version used in a full installation of MS Visual Studio.

In the Microsoft file *vsvars32.bat* file located in “Installing Microsoft Visual C++2005 Express Edition” on page 9

C:\Program Files\Microsoft Visual Studio 8\Common7\Tools

1. to the section [@set INCLUDE=] add the following path:
 - *C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include;*
2. to the section [@set LIB=] add the following path:
 - *C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Lib;*

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2005

Before you begin

Assumptions:

IBM DAV Client has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

IBM DAV Virtualizer has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

Microsoft Visual C++ 2005 has been installed.

Note: See “IBM DAV Installation Components” on page 6 for components installed by the above.

Configuring IBM DAV Virtualizer to interact with MS Visual Studio

About this task

If Microsoft Visual Studio C++ is installed to a directory other than *C:\Program Files\Microsoft Visual Studio 8* the following steps need to be performed:

1. Navigate to the IBM DAV Virtualizer install location (to the user preferred area specified in the installation of IBM DAV Virtualizer) and open the *preference.ini* file.

For Example:

C:\Program Files\IBM\IBM DAV\Virtualizer\davGen

2. Edit the entry
virtualizer.compiler.rootfolder.msvc = C:\Program Files\Microsoft Visual Studio 8
to point to the install location for C++.

Configuring IBM DAV Virtualizer for Microsoft Visual C++ 2003

Before you begin

Assumptions:

IBM DAV Client has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

IBM DAV Virtualizer has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

Microsoft Visual Studio 2003 has been installed.

Note: See “IBM DAV Installation Components” on page 6 for components installed by the above

Configuring IBM DAV Virtualizer to interact with MS Visual Studio 2003

About this task

By default Virtualizer is setup to work with MS Visual Studio 2005.

To use Virtualizer with MS Visual Studio 2003, change the following settings in the *virtualizerpreferences.ini* file:

- **Default settings**
 - `virtualizer.compiler.rootfolder.msvc= C:\Program Files\Microsoft Visual Studio 8`
 - `virtualizer.compiler.target.build= vc`
- **MS Visual Studio 2003 settings**
 - `virtualizer.compiler.rootfolder.msvc= C:\Program Files\Microsoft Visual Studio 7`
 - `virtualizer.compiler.target.build= vc7`

Configuring IBM DAV Virtualizer for Linux using GCC 4.1.2

Before you begin

Assumptions:

IBM DAV Client has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

IBM DAV Virtualizer has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

GCC 4.1.2 has been installed

Set the IBM_DAV_PATH and LD_LIBRARY_PATH

About this task

IBM_DAV_PATH is the path where IBM DAV is installed. By default this is `/usr/ibmdav/client`

Example

For Example:

IBM_DAV_PATH

```
export IBM_DAV_PATH=/usr/ibmdav/client
```

LD_LIBRARY_PATH

```
export LD_LIBRARY_PATH=$IBM_DAV_PATH/lib:$LD_LIBRARY_PATH
```

Chapter 5. IBM DAV Semantics

IBM DAV Comment Format

Semantics for the IBM DAV Library are specified by embedding comments into the header file of the user library.

Semantics for the IBM DAV Library are specified by embedding comments into the header file of the user library.

The format for DAV semantic comments is:

```
/**IBMDAV*  
  
...  
*/
```

These comments can be placed anywhere in the header file, but must surround each tag.

To view a sample semantic header file, see Appendix B, “Semantics - Sample Header File,” on page 53.

Tag Types

The following are the different types of IBM DAV semantics tags

1. “Library tag”
2. “Function tag” on page 16
3. “Function Tag: @param” on page 16
4. “Function tag @dimensions” on page 18
5. “Function tag @return” on page 18
6. “Struct tag” on page 18
7. “Void*” on page 19
8. “Strings” on page 20

Library tag

The IBM DAV library tag is an optional tag that can be used to provide additional information about the library. By default, IBM DAV invokes functions in a library with the same name as the library header file (i.e. *lib<header_name>.so*). The library tag can be used to specify an alternate DLL/SO name. This name will also be used in the name of the generated libraries. The generated libraries will be

Linux *lib<library_name>.so*

Windows *<library_name>.dll*

In addition to allowing the user to specify an alternate name, the library tag can be used to append a prefix/suffix to each IBM DAV generated function.

For example

```

/**IBMDAV* @library MyLibrary
    @ prefix dav_
    @ suffix _dav
    @lib_options "-lnuma -lspe2 -lm"
*/

```

This library tag executes functions in a user library called `MyLibrary.so` on the server and appends the prefix `dav_` as well as the suffix `_dav` to each generated function. If the user library contains a function called `function()`, IBM DAV would generate a function stub called `dav_function_dav()`. This example also illustrates the usage of the `@lib_options` tag that is used to pass additional linker options to the server-side generated code that are required by the user library.

Function tag

The format of an IBM DAV function tag is

```
@function name
```

The IBM DAV function tag must be used for each function that is to be DAV Virtualized, even if it doesn't require semantics. The function name specified by the tag must correspond to a function name.

An example of when semantics are required, is when a parameter is an array or a pointer.

Argument attributes must be defined for each function parameter that requires semantics. For all function members that require semantics, an `@param` tag is required.

Note:

If a header file is included in the library header file then its semantics must also be specified in the library header file.

For example

```

/**IBMDAV* @function nAddBasicLong
*/
long nAddBasicLong(long n1, long n2);

/**IBMDAV* @function func1 */
int func1(long arg1);

```

Function Tag: @param

When an argument requires semantics, an `@param` tag is required. The format of an `@param` tag is `@param[outputType]` where `[outputType]` is optional.

An example of when semantics are required for a parameter is when the parameter is an array or pointer.

The output attribute of the `@param` tag has one of three options:

- in** the parameter value will be copied to the server but not back again.
- inout** the value is sent to the server, then returned to the client machine.
- out** is used in the case of server-side allocated memory.

If this attribute is omitted the default attribute is assumed. The default attribute is **in**.

Examples:

Note: For information on the @dimension tag, see “Function tag @dimensions” on page 18

Example 1 @param (none specified :default value in)

If no attribute is specified for the @param tag the default value **in** is used, the parameter value will be copied to the server but not back again.

```
/**IBMDAV* @function max
    @param n1 @dimensions [10][20]
*/
short int max(short int n1[][20]);

/**IBMDAV* @function shnAdd2DArrayShortInt
    @param n1 @dimensions [10][20]
    @param n2 @dimensions [10][20]
*/
short int shnAdd2DArrayShortInt(short int n1[][20], short int n2[][20]);
```

Example 2 @param[in]

If the output attribute of the @param tag is set to **in**, the parameter value will be copied to the server but not back again.

```
/**IBMDAV* @function nAddArrayShort
    @param[in] n1 @dimensions [10]
    @param[in] n2 @dimensions [10]
*/
short nAddArrayShort(short *n1, short *n2);

/**IBMDAV* @function sumSquareMatrix
    @param[in] matrix @dimensions [N*N]
*/
double sumSquareMatrix( double* matrix, int N);
```

Example 3 @param[inout]

If the output attribute of the @param tag is set to **inout**, the parameter value is sent to the server, then returned to the client machine.

```
/**IBMDAV* @function reverseIntArray
    @param[inout] array @dimensions [size]
*/
int reverseIntArray( int* array, int size);

/**IBMDAV* @function funcInOut
    @param[inout] param1 @dimensions[n1*5]
*/
int funcInOut(double *param1, int n1);
```

Example 4 @param[out]

If the output attribute of the @param tag is set to **out**, the parameter value is used in the case of server-side allocated memory.

```

/**IBMDAV* @function fAddArrayFloat
    @param[out] n1 @dimensions [1][10]
    @param[out] n2 @dimensions [1][10]
*/
float fAddArrayFloat(float **n1, float **n2);

/**IBMDAV* @function funcReturn
@return @dimensions[*resultSize]
    @param[out] resultSize @dimensions[1][1]
*/
double* funcReturn(int **resultSize);

```

Function tag @dimensions

The @dimensions tag specifies the size of the variable that requires semantics. This allows you to specify an array size, for example, an array of size 20. The size of the variable can be specified statically or dynamically using a sub-set of C operators

@function name

For example

```

/**IBMDAV* @function nAddArrayInt
    @param n1 @dimensions [10]
    @param n2 @dimensions [10]
*/
int nAddArrayInt(int n1[], int n2[]);

/**IBMDAV* @function chAddArrayChara
    @param ch1 @dimensions [dim]
    @param ch2 @dimensions [dim]
*/
char chAddArrayChara(char ch1[], char ch2[], int dim);

```

Function tag @return

When a return type requires semantics, an @return tag is required.

An example of when semantics are required, is when the return type of the function is an array or a pointer

For example

```

/**IBMDAV* @function reverseArrayIntb
    @param [inout] arrayInt @dimensions [10]
    @return @dimensions [10]
*/
int* reverseArrayIntb(int *arrayInt, int a);

/**IBMDAV* @function nAddArrayConstSignedInta
    @param n1 @dimensions [dim]
    @param n2 @dimensions [dim]
    @return @dimensions [dim]
*/
const signed int *nAddArrayConstSignedInta(const signed int n1[],
const signed int n2[], int dim);

```

Struct tag

The format of the struct tag is

@struct name

The usage of the struct tag is similar to that of the @function tag. The @struct tag must be used to specify semantics for every struct for which DAV code should be

generated, even if the struct does not require semantics. The struct name specified by the tag must correspond to a struct/typedef struct name.

An example of when semantics are required is when a parameter is an array or a pointer.

Argument attributes must be defined for each struct parameter that requires semantics. For all struct members that require semantics, an @param tag is required.

For example,

```
/**IBMDAV* @struct myStruct
*/
typedef struct myStruct
{
    long lg;
    short sh;
    int x;
    float fl;
    double db;
    char ch;
    long long ll;
    long int li;
    short int shi;
    unsigned us;
}myStruct;

/**IBMDAV* @struct Details
    @param name @type string
*/
typedef struct structDetails
{
    char* name;
    int age;
    char sex;
}Details;

/**IBMDAV* @function copyStruct3
    @param[out] copyDetails @dimensions [1][10]
    @param[in] originalDetails @dimensions [10]
*/
void copyStruct3(Details** copyDetails, Details* originalDetails);
```

Void*

The @type tag must be used in the case of a void* to specify the type of the argument data. The type of the data is required in order to calculate the amount of data to be transferred. The @type tag can be used to specify the void* as any basic C type.

For example

```
/**IBMDAV* @function sum
    @param[in] array @dimensions [10] @type int */
public int sum(void* array, int size);

/**IBMDAV* @function nVariousVoidAdd
    @param arrayIn @dimensions [10] @type double
    @param[inout] arrayOut @dimensions [10] @type double
*/
int nVariousVoidAdd(void* arrayIn, void* arrayOut);
```

Strings

The @type tag can be used in the case of a char* to specify the char* as a NULL-terminated C string.

For example

The @type tag can be used in the case of a char* to specify the char* as a NULL-terminated C string.

```
/**IBMDAV* @function sum
    @param[in] str @type string */

public int length(char* str);
```

IBM DAV Dimension definitions

IBM DAV allows the user to specify the size of an array in several different ways. The size of an array can be specified using a variable or a number. The size of an array can also be specified using a number of C operators. The operators follow normal C rules and will be inserted into the generated code. It is the users responsibility to ensure that the operators are correct and that this operator will work for the various programming languages that IBM DAV supports (e.g. Java™).

Table 2. Supported C operators

C Operator	Description	Type
(expression)	Precedence	Scope/Indexing Operators
[expression]	Array Indexing	
+	Unary plus	Unary Operators
-	Unary minus	
!	Unary negation	
~	Bit-wise complement	
*	Pointer de-referencing	
*	Multiplication	Multiplication Operators
/	Division	
%	Modulus	
+	Addition	Addition Operators
-	Subtraction	
<<	Left-shift	Bitshift operators
>>	Right-shift	
<	Less Than	Comparison operators
<=	Less Than or Equal to	
>	Greater Than	
>=	Greater Than or Equal to	
&	Bit-wise AND	Bit-wise AND
^	Bit-wise XOR	Bit-wise XOR
	Bit-wise OR	Bit-wise OR
&&	Logical And	Logical And
	Logical Or	Logical Or

Table 2. Supported C operators (continued)

C Operator	Description	Type
(condition)? (if-case): (else-case)	Conditional	Conditional Operator

Chapter 6. IBM DAV Virtualizer Usage

This section provides a brief introduction to the usage of IBM DAV Virtualizer.

Before you begin

Assumptions:

IBM DAV Client has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

IBM DAV Virtualizer has been installed, see Chapter 3, “IBM DAV installation instructions,” on page 5.

See “IBM DAV Installation Components” on page 6 for components installed by the above.

About this task

Note: The header-file used as an example in this section is called *Library.h*, which is installed with the IBM DAV Client component. The default location for *Library.h* is

- **Windows** "C:\Program Files\IBM\IBM DAV\Client\samples\SampleLibrary\library\Library.h"
- **Linux** /usr/ibmdav/client/samples/SampleLibrary/library/Library.h

Note: By default the server-side library will be named after the header-file. In this example the user library will be called *libLibrary.so*. The user library must be generated with the same header-file, in this instance *Library.h*.

Note: The server installation comes prepackaged with the *libLibrary.so*.

Generating DAV Libraries

1. Cd to the Virtualizer DAVgen directory
Linux `cd /usr/ibmdav/virtualizer/davGen`
Windows `cd "\Program Files\IBM\IBM DAV\ virtualizer\davGen"`
2. To generate DAV libraries for the installed sample execute the following:
Linux `./sample.sh`
Windows `sample.bat`

General Usage is:

Linux `./davGen.sh --header=headerFile [--outputDir=dir]
[--preferences=properties]`

Windows `./davGen.bat --header=headerFile [--outputDir=dir]
[--preferences=properties]`

Note: When generating IBM DAV libraries the usage is case sensitive. If a parameter is incorrectly entered, the current directory will be use by default.

Table 3. General Usage for davGen.sh/ davGen.bat

Usage	Description
--outputDir	Output directory for the generated files. If no path is specified the current directory will be used
--header	Header file containing desired IBM DAV Virtualized functions
--preferences	Preferences file. If this file is not specified the default preferences file will be used

Results

For a successful generation the following output will be displayed

```
Validating header file: sample/Library.h
Header file validation successful
```

```
Parsing header file: sample/Library.h
Header file parse complete
```

```
Processing user semantics
Semantics processed successfully
```

```
Generating IBM DAV libraries
IBM DAV code generation successful
```

Sample Libraries

IBM DAV provides the following libraries:

- "BLAS"
- "LAPACK" on page 25
- "FFT" on page 26
- "Monte Carlo (MC)" on page 26

BLAS

About this task

BLAS (Basic Linear Algebra Subprograms)

BLAS is a set of libraries performing basic linear algebra operations on vectors and matrixes. For more information see the *Basic Linear Algebra Subprograms Library Programmer's Guide and API Reference* document.ument.

For Cell/B.E Optimized Libraries, **BLAS** sample source code, pre built libraries and samples are included with IBM DAV

BLAS files location

- **Windows** "C:\Program Files\IBM\IBM DAV\Client\samples\BLAS"
- **Linux** /usr/ibmdav/client/samples/BLAS

Note: The Cell SDK 3.0.0.3 (Cell SDK 3.0 fixpack 3) must be installed on the Cell BladeCenter/compute node(s).

Note: By default the necessary settings for BLAS are included in the `IBM_DAV.Conf` file. These settings are included to allow you to use the BLAS services in IBM DAV. To disable the BLAS services, comment out the related entries in the `IBM_DAV.Conf` file by prefixing the lines with `#`.

Note: To install BLAS for x86 perform the following :

```
yum install blas
yum install blas-devel
```

Note: To install BLAS on Cell BE follow the instructions provided by Cell SDK.

Note: To ensure that BLAS services are correctly loaded on an x86 node, SE Linux should not be set into enforcing mode. To do so you must modify the file `/etc/selinux/config` with the following: `SELINUX=disabled` or `SELINUX=permissive` then reboot the machine for this new setting to take effect.

LAPACK

About this task

LAPACK (Linear Algebra Package)

LAPACK is a software library that provides routines to solve systems of linear equations, least-squares solutions of the systems, eigenvalue problems, matrix decomposition, etc. For more information see the *LAPACK - Linear Algebra Package Library Programmer's Guide and API Reference* document.

For Cell/B.E Optimized Libraries, **LAPACK**, sample source code, pre built libraries and samples are included with IBM DAV

LAPACK files location

- **Windows** `"C:\Program Files\IBM\IBM DAV\Client\samples\LAPACK"`
- **Linux** `/usr/ibmdav/client/samples/LAPACK`

Note: The Cell SDK 3.0.0.3 (Cell SDK 3.0 fixpack 3) must be installed on the Cell BladeCenter/compute node(s).

Note: By default the necessary settings for LAPACK are included in the `IBM_DAV.Conf` file. These settings are included to allow you to use the LAPACK services in IBM DAV. To disable the LAPACK services, comment out the related entries in the `IBM_DAV.Conf` file by prefixing the lines with `#`.

Note: To install LAPACK for x86 perform the following :

```
yum install lapack
yum install lapack-devel
```

Note: To install LAPACK on Cell BE follow the instructions provided by Cell SDK.

Note: To ensure that LAPACK services are correctly loaded on an x86 node, SE Linux should not be set into enforcing mode. To do so you must modify the file `/etc/selinux/config` with the following: `SELINUX=disabled` or `SELINUX=permissive` then reboot the machine for this new setting to take effect.

Note: LAPACK functions are not supported in XLL, because the functions rely on one particular function that takes a structure as parameter. Since XLL does not support structure parameters, all functions in Lapack-Sobol cannot be called using XLL.

FFT

About this task

The FFT library is a collection of C routines for computing discrete Fourier transforms provided by the IBM Software Development Kit (SDK) for Multicore Acceleration Version 3.1.

The library implements several variants of FFT (Fast Fourier Transform) algorithms that are widely used in science and engineering. Currently, there is support for single precision one-dimensional (1D) FFT routines

For Cell/B.E Optimized Libraries, **FFT**, sample source code, pre built libraries and samples are included with IBM DAV

FFT files location

- **Windows** "C:\Program Files\IBM\IBM DAV\Client\samples\FFT"
- **Linux** /usr/ibmdav/client/samples/FFT

For more information see the *Fast Fourier Transform Library Programmer's Guide and API Reference* document.

Monte Carlo (MC)

Monte Carlo (MC) library, distributed with Cell SDK provides a set of tools to generate random number sequences and transform uniform distributions of the numbers into normal ones.

It includes seven types of Random Number Generators (RNGs)

- Hardware based,
- Kirkpatrick-Stoll,
- Mersenne Twister
- Mersenne Twister with Dynamic Creator
- SIMD - Oriented Mersenne Twister
- Sobol
- Sobol for High Dimensions

and three type of distribution transformation functions .

- Box-Mueller (Box-Mueller Cartesian - Sine only)
- Box-Mueller (Box-Mueller Cartesian - Sine and Cosine)
- Moro's Inversion
- Polar Method (Box-Mueller rejection method)
- Inverse Cumulative Distribution Function

For Cell/B.E IBM DAV provides an adapter library that allows Monte Carlo functions to be called directly from a remote client via DAV-enabled interface.

RNG files location

- **Windows** "C:\Program Files\IBM\IBM DAV\Client\samples\RNG"
- **Linux** /usr/ibmdav/client/samples/RNG

For more information see the *Monte Carlo Library Programmer's Guide and API Reference* document.

DAV Interface to Monte Carlo Library

IBM DAV supports all the features and algorithms available in the PPE (IBM PowerPC Processor Element) interface of Monte Carlo library available in the 3.1 version of the Cell SDK. The DAV interface mimics the original interface of the library as far as possible. However, some changes to the interface were necessary.

The DAV interface is defined in the `davrand.h` header file. Applications that want to access the functionalities of the Monte Carlo library using DAV should use this header file, and must be linked to the `libdavrand_oai.so` shared library in the client side. Both files are installed with the IBM DAV client binaries.

There are two main differences between the original interface and the interface provided by DAV. First, instead of using `mc_rand_add_operation()`, there is one separate function for adding each RNG and each transformation. The structures for passing the parameters to the algorithms are the same as originally defined in the Monte Carlo library. Second, instead of using `mc_rand_perform()`, there is one function for each data type being returned (unsigned int, float or double).

Table 4. How the Original Interface is accessed using DAV

Original Interface	DAV Interface (substitute for Original Interface)
<code>mc_rand_initialize</code>	<code>mc_rand_init</code>
<code>mc_rand_add_operation</code>	<code>mc_rand_add_mt ()</code> <code>mc_rand_add_dcmt ()</code> <code>mc_rand_add_sfmt ()</code> <code>mc_rand_add_ks ()</code> <code>mc_rand_add_sb ()</code> <code>mc_rand_add_sb2 ()</code> <code>mc_rand_add_hw ()</code> <code>mc_rand_add_po ()</code> <code>mc_rand_add_bm ()</code> <code>mc_rand_add_bm2 ()</code> <code>mc_rand_add_mi ()</code>
<code>mc_rand_perform</code>	<code>mc_rand_perform_u4()</code> <code>mc_rand_perform_f4()</code> <code>mc_rand_perform_d2()</code>
<code>mc_rand_terminate</code>	<code>mc_rand_end</code>

Sobol and HD Sobol parameters

When adding Sobol and HD Sobol operations, note that the corresponding add operation functions have extra parameters for passing a pointer to the initialization table (called directions table). The pointers to the table must be passed using these parameters and not the structure. For HD Sobol, the first four members of the parameters structure, which are optional and can be used to allocate PPE memory for temporary use, are ignored by DAV. Instead, an extra parameter, called `doMalloc`, can be used to specify if PPE memory should be allocated for the generation of random numbers.

Library Semantics

About this task

In order to remotely execute a user function, IBM DAV must be able to determine the amount of data that must be sent to the server in order to correctly execute the user library. IBM DAV can determine the correct way for transporting basic C datatypes such as short, long, int, and float. However, IBM DAV requires more information to transport more complex C data types (arrays, pointers, and structs).

For example, IBM DAV must know the number of elements in an array. Semantics provide a method for the user to specify this information.

Semantics for the IBM DAV library are specified by embedding custom comments into the header file of the user library. For more information on IBM DAV semantics see Chapter 5, "IBM DAV Semantics," on page 15

Chapter 7. Semantic Verification

About this task

Semantic verification is the process of verifying the correctness of the semantic inputs which were added to the application header file.

For Example: If the dimension of an array is specified by another parameter, we must verify that the correct parameter has been specified. IBM DAV Virtualizer provides a component, the semantic verifier, to assist with this process.

Semantic verification is performed entirely on the local machine, thereby eliminating any network issues that might otherwise complicate the process.

1. Add your native library to your system library path

Linux `lib%LibraryName%.so`

Windows `%LibraryName%.dll`

Note:

Windows To create the sample library *Library.dll*

- Open the Library project in
`C:\Program Files\IBM\IBM DAV\Client\samples\SampleLibrary\library`
- Build the DLL in Release mode
- Add this DLL to the system path. **For Example**, this DLL could be placed in
`C:\Program Files\IBM\IBM DAV\Client\bin`

Linux To create the sample library *libLibrary.so*

- `cd /usr/ibmdav/client/samples/SampleLibrary/library`
 - `make -f makefile`
 - Add this library to your LD_LIBRARY_PATH
2. Run the **Windows** *sample.bat* / **Linux** *sample.sh* in the virtualizer. Open the *LibraryTester.cpp* in **Windows** `C:\program files\IBM\IBM DAV\Virtualizer\output\client` / **Linux** `/usr/ibmdav/virtualizer/output/client`. The following test methods are generated:

```
void test_calculate_Array()
{
}

void test_calculate_Array2()
{
    //Your function is now IBM DAV enabled
    //You can use your function with IBM DAV by using
    //calculate_Array2( arg1, arg2, arg3 );
}

void test_reverseIntArray()
{
    //Your function is now IBM DAV enabled
    //You can use your function with IBM DAV by using
    //reverseIntArray( arg1, arg2 );
}
```

```

void test_matrixDeterminant()
{
    //Your function is now IBM DAV enabled
    //You can use your function with IBM DAV by using
    //matrixDeterminant( arg1, arg2, arg3 );
}

void test_sumSquareMatrix()
{
    //Your function is now IBM DAV enabled
    //You can use your function with IBM DAV by using
    //sumSquareMatrix( arg1, arg2 );
}

```

3. Add the following to each of the methods:

```

void test_calculate_Array()
{
    int size = 3;
    double *arg1=new double[3];
    double *arg2=new double[3];
    for(int i=0; i<size; i++)
        arg1[i]=i;

    double result = calculate_Array( arg1, arg2, size );
    std::cout << "calculate_Array():\n";
    std::cout << "Expected Result is:\t6\n";
    std::cout << "Actual Result is:\t" << result;
    std::cout << "\n";
}

void test_calculate_Array2()
{
    int size = 3;
    double arg1[] = {0.0, 1.0 , 2.0};
    double arg2[3];

    double result = calculate_Array2( arg1, arg2, size );
    std::cout << "calculate_Array2():\n";
    std::cout << "Expected Result is:\t6\n";
    std::cout << "Actual Result is:\t" << result;
    std::cout << "\n";
}

void test_reverseIntArray()
{
    int *array = new int[5];
    for(int i=0; i<5; i++)
        array[i] = i;

    reverseIntArray(array, 5);

    std::cout << "reverseIntArray():\n";
    std::cout << "Expected Result is:\t4 3 2 1 0\n";
    std::cout << "Actual Result is:\t";

    for(int i=0; i<5; i++)
        std::cout << array[i] << " ";

    std::cout << "\n";
}

void test_matrixDeterminant()
{
    double matrix[2][20];
    matrix[0][0] = 1;
    matrix[0][1] = 0;
}

```

```

matrix[1][0] = 0;
matrix[1][1] = 1;

std::cout << "matrixDeterminant():\n";
std::cout << "Expected Result is:\t1\n";
std::cout << "Actual Result is:\t";
std::cout << matrixDeterminant(matrix, 2, 2);
std::cout << "\n";
}

void test_sumSquareMatrix()
{
    int N = 3;
    double *matrix = new double[N*N];
    for(int i=0; i<N*N; i++)
        matrix[i] = 1;

    std::cout << "sumSquareMatrix():\n";
    std::cout << "Expected Result is:\t9\n";
    std::cout << "Actual Result is:\t";
    std::cout << sumSquareMatrix(matrix, N);
    std::cout << "\n";
}

```

4. From the command line run the script **Windows** (run.bat)/ **Linux** (run.sh) and ensure the expected results and actual results are the same

Chapter 8. Server Side Deployment

Before you begin

Assumptions:

- For Server-side deployment you must have write access to the install directory.
- The relevant IBM DAV server rpm has been downloaded and installed, see “Server-side installation for Linux” on page 7.
 - *dav-server.i386.rpm* for Red Hat Linux Server 5.1, 5.2
 - *dav-server.ppc64.rpm* for Cell/B.E processors, IBM BladeCenter® QS21, QS22.

Note: This is installed automatically to */usr/ibmdav*.

To use an alternative location copy */usr/ibmdav* to the new location and update the `IBM_DAV_PATH`

- The server-side libraries have been built.
- The sample server-side application library *libLibrary.so* has been provided.
- The IBM DAV Deployer requires that the server `IBM_DAV_PATH` is set in the users `.bashrc` file.

Automatic Deployment

IBM DAV Virtualizer includes an automated deployment tool that can be used to deploy IBM DAV libraries. This program uses the server files generated by the Virtualizer to prepare the IBM DAV library for deployment.

The Deployer includes the following functionality

- Configurable to determine build and deployment machines.
- Copies generated files to build machine and builds the `lib%LibraryName%_oai.so` library.
- Deploys Library on each machine listed in configuration file

Before the deployer script can be run, the user must enter node details. This file can be found in the deployer directory of the virtualizer:

ConnectionDetails.txt :

```
user1:pass1@buildHost1:port1
user2:pass2@deployHost1:port2
```

- Line 1 represents the node on which the library will be built
- Line 2 represents the node on which the library will be deployed
- The User can specify more than one location in configuration file to enable multiple deployments

Note: The build machine and deployment machine can be one machine. If different machines, they must have the same platform, architecture and OS

Once connection details have been added, the deployer builds and deploys libraries. It can be invoked from the command prompt:

deployer --dir=./GeneratedCode/server --userlist=connectiodetails.txt --scriptArgs /headerPath /libraryPath port

Table 5. General Usage *deployer.sh/ deployer.bat*

Usage	Description
dir:	Location of Virtualizer generated server files. Optional. The current working directory will be used by default
userlist:	Text file containing details for connecting to remote machines via SSH with one entry per line. Connection details are to be in the form user[:password]@host[:port]. The first entry indicates the build machine while additional entries specify deployment machines. At least 2 entries are needed. The same host can be used as a build and deployment machine. Note: Users specified for deployment machines require write access on \$IBM_DAV_PATH path (i.e. by default IBM DAV is installed by root)
scriptArguments:	Arguments to be passed to the build and deploy scripts in addition to the library name.
headerPath:	Path to the library header file on the build machine.
libraryPath:	Path to the <i>lib%LibraryName%.so</i> file on the deployment machine.
port:	Port that the server will use to listen for incoming DAV Requests

During the deployment process, updates will be made to the server-side *IBM_DAV.conf* file, located in */usr/ibmdav/server* to reflect the entries in the virtualizer-generated *changes_to_server_config_file.txt*

Chapter 9. Running the Server-side IBM DAV Service

Server-side IBM_DAV.conf

The server side of the IBM DAV Server uses the configuration file *IBM_DAV.conf*.

- By default the *IBM_DAV.conf* file is located in *\$IBM_DAV_PATH*
- The logfile location can be set by changing *dav.log.directory* and *dav.log.filename* in the *IBM_DAV.conf*.

For other server-side configuration options, see Appendix C, “Server Side IBM_DAV.conf settings,” on page 57.

Start the Service Broker

About this task

The Service Broker:

- Logs services and requests running on the compute nodes.
- Accepts client requests for a service and directs the client to an appropriate available service
- Can run on a separate node or on one of the compute nodes.

Note: In the case where the Service Broker and services are on the same machine, they must be started as the same username.

Start the Service Broker located from *\$IBM_DAV_PATH/bin* using one of the following

- *./davServiceBroker*
Starts the Service Broker as a daemon
- *./davServiceBroker -s*
Stops the Service Broker

Launch the Service

About this task

An IBM DAV Service is a process which will accept off-loaded function-calls from an IBM DAV client

The service accepts requests for the set of functions provided by the server-side Library loaded into this Service type.

The service listens for client connections which have been directed to it by the Service broker.

Start the IBM DAV Service(s) listed in the *IBM_DAV.conf* by calling the daemon located in *\$IBM_DAV_PATH/bin*

- **For Example:** *\$IBM_DAV_PATH/bin/dav start*

Verify the service is running using 'dav status'

The user, service-name and the port that the service is running on are shown by calling

- **For Example:** `$IBM_DAV_PATH/bin/dav status`

What to do next

See Appendix C, "Server Side IBM_DAV.conf settings," on page 57 for additional server-side IBM_DAV.conf options.

Stopping the IBM DAV Service

To stop all services call

- **For Example:** `$IBM_DAV_PATH/bin/dav stop`

What to do next

See Appendix C, "Server Side IBM_DAV.conf settings," on page 57 for additional server-side IBM_DAV.conf options

Running a Service - Dedicated Mode

About this task

A server-side option is available in the configuration file to allow a service to be run in dedicated mode. This feature has particular performance benefits for the Cell BE architecture.

In dedicated mode a service will run a client's request exclusively in the main program. No other requests can run concurrently for that particular service.

To enable dedicated mode for a service, the following line must be listed in the server-side configuration file for the service.

For Example for the Library sample:

- `dav.server.services.Library.dedicated=1`

Setting an alternative location for the shared object

- The IBM_DAV.conf file allows the user to set the absolute path of the IBM DAV service to any preferred location using:
 - `dav.server.service.<service_name>.root`
 - **For Example:** `dav.server.service.Library.root=/usr/lib`
- A path relative to the `$IBM_DAV_PATH` directory is also possible using the parameters `dav.server.services.root` and `dav.server.service.<service_name>.root`. In the case of the sample application the following settings are already in the IBM_DAV.conf file:
 - `dav.server.services.root=services`
 - `dav.server.service.Library.root=Library`
- The dav executable will then look in the `$IBM_DAV_PATH/services/Library` directory for the dependent shared objects.

- To run the sample service using the above settings move *libLibrary.so* and *libLibrary_oai.so* from the *\$IBM_DAV_PATH/bin* to the *\$IBM_DAV_PATH/service/Library* directory and restart the service.
- If the *dav.server.service.root* entry is missing in *IBM_DAV.conf*, its value defaults to *\$IBM_DAV_PATH/bin*.

What to do next

See Appendix C, “Server Side IBM_DAV.conf settings,” on page 57 for additional server-side IBM_DAV.conf options

Chapter 10. IBM DAV Virtualizing the Client Application

About this task

The Client Application is IBM DAV Virtualized when it is configured to link with the generated IBM DAV Virtualizer libraries generated in “Generating DAV Libraries” on page 23

This enables the Client Application to execute calls to the remote IBM DAV Service.

This section describes how to carry out the IBM DAV Virtualizing of the Client Application for Linux and Windows.

Virtualizing a C++ Client Application on Linux

Before you begin

Assumptions:

The IBM DAV Client on Linux has been installed.

The IBM DAV Virtualizer on Linux has been installed and used to generate code.

Server side application library has been prepared.

IBM_DAV_PATH is set to */usr/ibm/client*

About this task

To Virtualize a client application through Linux

1. Add the *_oai.so* file (and *\$IBM_DAV_PATH/lib*) to the system library path.
2. Create a makefile that links to the **_oai.so* file

Note: For details on creating this makefile see Appendix F, “makefile for IBM DAV Virtualizing a Linux application,” on page 63.

3. Run **make** to build the IBM DAV-Virtualized executable
 - **make -f makefile**
4. Set the correct values in your *IBM_DAV.conf* then run the generated executable file.

Virtualizing a C++ Client Application on Windows

Before you begin

Assumptions:

The IBM DAV Client has been installed.

The IBM DAV Virtualizer has been installed.

The IBM DAV Generator has been run, and libraries created.

About this task

To Virtualize a client application through Windows, carry out the following tasks:

Startup

About this task

Note: This section refers to MS Visual C++ 2005 (or MS Visual C++ 2005 Express Edition).

Note: For the purposes of the document screenshots refer to Microsoft Visual C++ 2005 Express Edition.

1. Open MS Visual C++ 2005 (or MS Visual C++ 2005 Express Edition if this was installed in Chapter 4, "IBM DAV Virtualizer Configuration," on page 9).
2. Open the client application which is to be IBM DAV Virtualized or if following the sample application, create a new project 'Sample' by selecting from the menu bar
 - **File**→**New**→**Project**
3. Choose **Win32** in the Project types pane and **Win32 Console Application** in the Templates pane and set the Project name (**For Example: Sample**)
4. On the following screen, click **Next**.
5. Uncheck the box **precompiled header** in the 'Additional options' section of the next screen and then click **Finish**.
6. For the sample application after the project has created, add the *LibraryDriver.cpp* file by right clicking on Source Files. Select **Add**→**Existing Item**
7. Browse to the sample application c folder and select *LibraryDriver.cpp*. By default this is in the folder:
 - `"C:\Program Files\IBM\DAV\samples\SampleLibrary\c"`.

Set additional lib path

1. Right click your project and select **Properties**
2. Set the linked library path:
 - **Project**→**Properties**→**Configuration Properties**→**C/C++**→**General**→**Additional Library Directories**
3. Add the client directory created by the IBM DAV Virtualizer in "Generating DAV Libraries" on page 23, click **Apply** then click **OK**

Note: for the dll's created during the IBM DAV Virtualizer step, *Library_oai.dll* generation must be either copied to the same location as the executable or be in the PATH environment variable. The default location for the sample application dll - *Library_oai.dll* is: `c:\Program files\IBM\IBM DAV\Clientsamples\SampleLibrary\GeneratedCode\client`

Set additional Include path

1. Right click your project and select **Properties**
2. Set the include header file path:
 - **Project**→**Properties**→**Configuration Properties**→**C/C++**→**General**→**Additional Include Directories**

3. Add the client directory created by the IBM DAV Virtualizer in “Generating DAV Libraries” on page 23, click **Apply** then click **OK**

Add the IBM DAV generated library to the dependencies

Add the library which was generated by the IBM DAV Virtualizer in “Generating DAV Libraries” on page 23

- **Project>Properties>Configuration Properties>Linker>Input>Additional Dependencies**

Note: The **Apply** button must be clicked to submit the changes before clicking **OK**

Build the Client Application

Build the project to produce the client exe file. The application is now IBM DAV Virtualized.

- Set the linked library path:
 - **Project>Properties>Configuration Properties>Linker>General>Additional Library Directories**
- Add the client directory created by the IBM DAV Virtualizer in “Generating DAV Libraries” on page 23, click **Apply** then click **OK**

Note: The default location for the sample application dll - *Library_oai.dll* is:
`C:\Program Files\IBM\IBM DAV\Virtualizer\davGen\output\client`

Virtualizing an Excel Client Application

Before you begin

Assumptions:

The IBM DAV Client has been installed.

The IBM DAV Virtualizer has been installed.

The IBM DAV Generator has been run, and library created.

About this task

To virtualize an MS Excel/VBA application through Windows, carry out the following steps:

StartUp

1. Open MS Excel.
2. Open the client application to be Virtualized, if following the sample application, then open **LibraryDriver.xls**. By default this is in the folder:
`C:\Program Files\IBM\IBM DAV\Ckient\samples\SampleLibrary\excel`
3. When prompted select **Enable Macros**.

Replace references to local library with references to Virtualized library

1. Open the Visual Basic Editor by selecting **Tools>Macro>Visual Basic Editor**
2. On the VBA Project Navigation pane select **Modules>Module 1**.

3. Locate the Declare statements that reference the functions from the **Library.h** file as in Appendix A, "Sample Header File," on page 51
4. Change the Lib section of the Declare statement to reference the library created by the IBM DAV Generator in "Generating DAV Libraries" on page 23
5. Change the Alias section of the Declare statement to accommodate the VBA specific entry-points to the functions in the library created by the IBM DAV Generator in "Generating DAV Libraries" on page 23

For Example: Alias **calculate_Array** becomes Alias **calculate_Array_VBA**

6. Repeat Steps 4. and 5. for each of the functions that you wish to offload to the server.
7. The Declare statement should now look like this:

```
Declare Function Library_calculate_remote Lib "Library_oai.dll" Alias  
"calculate_Array_VBA" (ByRef array1 As Double, ByRef array2 As  
Double, ByVal value As Long) As Double
```

Your MS Excel spreadsheet is now enabled

Virtualizing a Java Client Application

Before you begin

Assumptions:

The IBM DAV Client has been installed.

The IBM DAV Virtualizer has been installed.

The IBM DAV Generator has been run, and libraries created.

About this task

To virtualize a Java application: carry out the following tasks:

1. In your **Driver** you need this import to handle exceptions thrown by the <library_name>Client object:

```
import com.ibm.mathsci.mshcalc.MshException;
```
2. Then write your **Driver** as you would normally. Create a new Client object, and call your functions as functions of the object. The function names are the same as in the header file.
3. To compile, add the **davCore.jar** to your classpath:

```
javac -cp "%IBM_DAV_PATH%\lib\davCore.jar" <package_name>\*.java
```
4. To run from command line:

Note: If your working environment is the GNU/Linux OS, substitute the ; with :

Chapter 11. Running the client application

Before you begin

Assumptions:

- **Windows**

- The IBM DAV client has been installed and has set the `$IBM_DAV_PATH`.
- The default value is `C:\Program Files\IBM\IBM DAV\Client`.
- `$IBM_DAV_PATH/bin` is appended to the sys path which enables the client to communicate with the active server-side IBM DAV service using the `IBM_DAV.conf`.

- **Linux**

- The IBM DAV client has been installed, but it sets no env variables.
- Each time you run the client, you must have the following set (i.e. for the default installation)
 - **export** `IBM_DAV_PATH=/usr/ibmdav/client/`
 - **export** `LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/ibmdav/client/lib`

Note: These can be added to your `.bashrc` file.

Update the server settings in the client-side `IBM_DAV.conf`

About this task

Before running the Client Application the `IBM_DAV.conf` must be edited to allow Client communication with the IBM DAV Service.

1. The `IBM_DAV.conf` can be found under the `$IBM_DAV_PATH` directory.
2. The following parameters must be updated:

<code>dav.servers</code>	This is the Hostname of the server. For Example: <code>dav.servers=myserver.</code>
<code>myserver.dav.server.port</code>	This is the port the server is running on For Example: <code>myserver.dav.server.port=4320</code> This must be set to the same value as set in the <code>IBM_DAV.conf</code> file on the server itself, or the port set by the DAV daemon.
<code>myserver.dav.server.ip</code>	This is the port the server is running on For Example: <code>myserver.dav.server.port=4320</code> This must be set to the same value as set in the <code>IBM_DAV.conf</code> file on the server itself, or the port set by the DAV daemon.

Note: It is assumed that a Service broker will be used otherwise a service IP Address and port must be entered.

3. A logfile can be specified in the IBM_DAV.conf to see execution messages using the parameters dav.log.directory and dav.log.filename
 - **For Example Windows:**
 - dav.log.directory=C:\
 - dav.log.filename=ibm_dav.log
 - **For Example Linux:**
 - dav.log.directory=/home/user/
 - dav.log.filename=ibm_dav.log

What to do next

For other client-side configuration options see Appendix D, “Client Side IBM_DAV.conf settings,” on page 59.

Launch the Application

1. Run the Client Application. This is located under the project directory created in Chapter 10, “IBM DAV Virtualizing the Client Application,” on page 39
2. For the sample application the default location is the release directory under:
 - **Windows:**
 - *C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Projects*
 - **Linux:**
 - Use the path you created in “Virtualizing a C++ Client Application on Linux” on page 39

Running an Excel application

1. Copy the IBM DAV GEN-erated library (_oai.dll) to the same directory as your executable.
2. Create a BATCH file to tailor PATH for application on Windows
A Batch file will temporary change your systems PATH. Create a new file and add the following:
 - • To run an MS Excel application


```
@echo off
%~d0
cd "%~p0"
set OPATH=%PATH%
set PATH=%IBM_DAV_PATH%\bin;C:\yourLibraryLocation;%PATH%

cmd /c start Excel <ExcelSpreadSheetName>.xls
set PATH=%OPATH%
```

Running a Java application

About this task

The generated Java sources and the davCore.jar are the only libraries required to send Java requests to IBM DAV You have to include davCore.jar in the CLASSPATH. Your directory structure must match the package structure that the Java application expects.

- **Windows**

Add to the batch file, or call manually in Command Prompt, from one directory above your source files:

```
javac -cp "%IBM_DAV_PATH%\lib\davCore.jar" <package_name>\*.java  
java -cp "%IBM_DAV_PATH%\lib\davCore.jar";. <package_name>\.Driver
```

- **Linux**

Add the following to script file, or call manually from shell:

```
javac -cp "$IBM_DAV_PATH\lib\davCore.jar" <package_name>\*.java  
java -cp "$IBM_DAV_PATH\lib\davCore.jar";. <package_name>\.Driver
```

Note: If you are sending large amounts of data to a DAV server in Java you need to add an extra command when you call the program. The Java Virtual Machine has a default heap size, on 32bit systems the maximum is about 64Mb. When offloading computationally intensive functions to a server this limit is often too small. You can increase the heap size with the argument: `-Xms32m -Xmx256m`. This sets the initial heap size to 32Mb, and the maximum allowed size to 256Mb.

```
java -cp "%IBM_DAV_PATH%\lib\davCore.jar";. -Xms32m -Xmx256m  
<package_name>.Driver
```

Chapter 12. XLL Add-ins

In an Excel spreadsheet it is possible to call some built-in functions directly from a spreadsheet cell using the operand: `=my_function(...)`, such as the built-in `=sum()`, `=max()`.

These functions take arguments that can be given by value (**For Example:** `max(2,42)`) or by cell reference (**For Example:** `max(42,A1)`).

Excel allows the definition of new functions that will be accessible from the spreadsheet through XLL Add-ins

The XLL Add-ins are the fastest for data processing as they interact directly with the Excel core.

Note: XLL Add-ins are configured to work with **Microsoft Visual C++ 2005(8)**

Note: For more information on XLL Add-ins, see the MSDN Excel 2007 XLL Software Development Kit Documentation webpage at:

- <http://msdn.microsoft.com/en-us/library/bb687883.aspx>

Enabling XLL Add-in Functionality

About this task

To enable and use XLL Add-in Functionality, carry out the following steps:

1. Download and install Excel 2007 SDK from the Excel 2007 XLL Software Development Kit webpage.
 - <http://www.microsoft.com/downloads/details.aspx?FamilyId=5272E1D1-93AB-4BD4-AF18-CB6BB487E1C4&displaylang=en>
2. Open the IBM DAV Virtualizer preferences.ini. located at `c:\Program Files\IBM\IBM DAV\Virtualizer\davGen`
3. Uncomment the entry `virtualizer.compiler.rootfolder.msxml` and update the path of the SDK, if necessary.
4. Run the Virtualizer (for more details see Chapter 10, "IBM DAV Virtualizing the Client Application," on page 39) , to generate the `*libraryname*.xml` in the client directory.

XLL Use

- Once generated the XLL can be used on any machine with the IBM DAV Client installed.
- An IBM DAV generated XLL is compatible with the following versions of Excel: 2002, 2003 and 2007.
- Any error reported during the execution (from the Client, Service Broker or Server) is handled by the XLL. The resulting cell will contain the corresponding error message starting with `#DAV!ERR:`

Note: If the provided data during a function call is not as expected (**For Example**, array size is not as defined in the semantic), the XLL will return an error message starting with #DAV!ERR: followed by the actual description of the problem.

- By default Excel shows only the result in the calling cell . To extend the result area to display further information (**For Example** in case of a returned array), you need to select the area, return to the **Formula Bar** and press **Ctrl+Shift+Enter**.

Note: All arguments specified in the semantic as [inout] or [out] will be returned in formatted array following the result. To display them use the **Ctrl+Shift+Enter** as described above.

Using XLL Add-in Functionality - Example

Before you begin

If you are not are not connected to your server, this process will fail.

To add the Library.xll to Add-ins in Excel

1. Open Excel, to use an XLL add-in:
 - **Excel 2003**
 - Go to **Tools -> Add-ins**
 - **Excel 2007**
 - Click the Office Button, then click **Excel Options** or press **Alt+FI** to open the Excel Options dialog box and then click the **Add-Ins** tab.

Note: A sample library.xll file is included at C:\Program Files\IBM\IBM DAV\Client\samples\SampleLibrary\GeneratedCode\client

2. In the Add-Ins dialog box click **Browse**, search for and select the *.xll file located at C:\Program Files\IBM\IBM DAV\Client\samples\SampleLibrary\GeneratedCode\client, then click **OK**. The add-in is then displayed in the Add-ins dialog box with its corresponding check box selected. Click **OK** to return to your spreadsheet.
3. You can now use your Dav functions like any other Excel function. These are included in IBM DAV *Libraryname*

About this task

Using your IBM DAV functions

1. Open the provided LibraryDriverXll.xls excel sample file, this is located at C:\Program Files\IBM\IBM DAV\Client\samples\SampleLibrary\excel
2. The excel file demonstrates the calculate_array function
 - The **in** values are A4:A8
 - The **out** values are B4:B8
 - The **size** value(5) is in C4
 - The **Result** value(30) uses the function {=calculate_Array(A4:A8,B4:B8,C4)}
 - The **Calculate_Array::out** values are D7:D11
3. By updating the **in** cells, Excel will communicate with DAV through the XLL and automatically update the **Result** section with new values.

4. To add select a new function
5. Select a new cell to view your XLL Add-in calculated results
6. Access the XLL Add-in
 - **MS Excel 2003** - Click **Insert/Function**, click **Fx** or **Shift-F3** to open the **Insert Function** window
 - **MS Excel 2007** - Click **Fx** or **Shift-F3** to open the **Insert Function** window
7. In the select a category dropdown menu select **IBM DAV Library**
8. Select the required XLL Add-in from the **Select a function** list then click **OK**.
9. Use the Function Arguments menu to Enter or Select the required cells for processing then click **OK**

Note: Values can be either entered as a number in the Function Arguments menu or can point to a specific cell.

10. The result of the processed calculation will appear in the cell selected in Step 5. As data within the fields selected in Step 9 is changed the result in the function field cell selected in Step 5 will automatically change.

XLL Limitations

These limitations apply to any version of excel.

For issues in Excel XLL Development, see the Known Issues in Excel XLL Development webpage at:

- <http://msdn.microsoft.com/en-us/library/bb687841.aspx>

Because Excel is not a development tool, there is no support for complex programming operations. The following are not supported

- Structures
- More than 2 dimensional arrays (Excel spreadsheets are only 2 dimensional arrays)
- Modification in place of arguments

The following are limitations:

- Dimension expressions (see “IBM DAV Dimension definitions” on page 20) can only include single values, scalar variables, and one-dimension arrays. Expressions including two-dimension arrays and higher are not supported. **For Example:**

```
/**IBMDAV* @function function
    @param[inout] toto @dimensions [2]
    @param[inout] lapin @dimensions [5][toto[0]]
    @param[inout] lapin2 @dimensions [5][lapin[0][0]]
*/
void function(int* toto, short** lapin, short** lapin2);
```

In this example, the semantics for lapin are valid, but the semantics for lapin2 are invalid for XLLs, since one of its dimensions is specified using a two-dimension array.

- Strings are automatically truncated if their length exceeds 255 characters.
- Functions taking enum as arguments are supported, however the enum type is replaced by an 'int' argument. So when calling such a function, it is necessary to specify the numeric value instead of its name within Excel.

- Type long long int (64 bits) is not natively supported by Excel, thus they will be transposed as double to be used. This can lead to accuracy loss.

Note: If a function is not supported by an xll, it is not added in the list of methods and can not be accessed from Excel

XLL Assumptions

The followings are the assumptions for XLL Add-ins

1. When an array of numbers is expected as an input argument, any strings in the selected array will be considered as value **0.0**
2. When an array of numbers is expected as an input argument, any empty cell is considered as **0.0**
3. **char type** are treated as pure numbers within Excel, thus it is not possible to use C `ascii` value such as 'A' or 'B'; their decimal value needs to be used instead (**65, 66**).
4. Native data type in Excel is double, but functions taking other data types are exported and made available from Excel. You must be careful of **data overflow** when providing a value that exceeds the range of the expected type. There is no mechanism to check data consistency so the user must provide valid data.
For Example: when an unsigned char [0-255] is expected and the given value is 258 (greater than 255) the value provided to the exported function will be viewed as (unsigned char)258, thus resulting to value 3.
5. Function name collisions: if a function is exported more than once (by different XLLs), Excel behavior remains uncertain, and may crash. To prevent this, never have 2 versions of the same XLL loaded at the same time.
6. Excel cannot properly unload an XLL: it will only make the function call invalid by specifying no argument for it. To completely unload an XLL, excel needs to be restarted. In order to reload a new XLL version, do the following steps:
 - Select **Menu: Tools->Add-ins**
 - Uncheck the XLL to unload, then click on **OK** to close the dialog box
 - Select **Tools->Add-ins** then select the required XLL...
7. One Dimensional arrays can be selected on a single line or a single column.
8. Two Dimensional arrays are line-wise represented in excel (`my_array[nb_lines][nb_columns]`).
9. All functions exported in an XLL, have to return a value. So exported void functions will return the default value **True**

Appendix A. Sample Header File

Sample header file Library.h

```
//  
// Licensed Materials - Property of IBM  
// Restricted Materials of IBM  
// 10604 - IBM Dynamic Application Virtualization.  
// Copyright IBM Corp. 2007, 2008 All Rights Reserved.  
// US Government Users Restricted Rights  
// - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with  
// IBM Corp.  
//  
  
#if defined(__cplusplus)  
extern "C" {  
#endif  
  
/**IBMDAV* @function calculate_Array  
    @param[in] in @dimensions [size]  
    @param[inout] out @dimensions [size]  
    */  
double calculate_Array(double *in, double *out, int size);  
  
/**IBMDAV* @function calculate_Array2  
    @param[in] in @dimensions [size]  
    @param[inout] out @dimensions [size]  
    */  
double calculate_Array2(double in[], double out[], int size);  
  
/**IBMDAV* @function reverseIntArray  
    @param[inout] array @dimensions [size]  
    */  
int reverseIntArray( int* array, int size);  
  
/**IBMDAV* @function matrixDeterminant  
    @param[inout] matrix @dimensions [x_size]  
    */  
double matrixDeterminant( double matrix[][20], int x_size, int y_size);  
  
/**IBMDAV* @function sumSquareMatrix  
    * @param[in] matrix @dimensions [N*N]  
    */  
double sumSquareMatrix( double* matrix, int N);  
  
#if defined(__cplusplus)  
};  
#endif
```

Appendix B. Semantics - Sample Header File

```
#if defined(__cplusplus)
extern "C" {
#endif

/***** 1) Library Tag *****/
/**IBMDAV* @library MyLibrary
    @ prefix dav_
    @ suffix _dav
    @lib_options "-lnuma -lspe2 -lm"
*/

/***** 2) Function Tag *****/
/**IBMDAV* @function nAddBasicLong
*/
long nAddBasicLong(long n1, long n2);

/**IBMDAV* @function func1 */
int func1(long arg1);

/***** 3) Function Tag: @param *****/
/**IBMDAV* @function max
    @param n1 @dimensions [10][20]
*/
short int max(short int n1[][20]);

/**IBMDAV* @function shnAdd2DArrayShortInt
    @param n1 @dimensions [10][20]
    @param n2 @dimensions [10][20]
*/
short int shnAdd2DArrayShortInt(short int n1[][20], short int n2[][20]);

/***** 4) Function Tag: @dimensions *****/
/**IBMDAV* @function nAddArrayInt
    @param n1 @dimensions [10]
    @param n2 @dimensions [10]
*/
int nAddArrayInt(int n1[], int n2[]);

/**IBMDAV* @function chAddArrayChara
    @param ch1 @dimensions [dim]
    @param ch2 @dimensions [dim]
*/
char chAddArrayChara(char ch1[], char ch2[], int dim);

/**IBMDAV* @function nAddArrayShort
    @param[in] n1 @dimensions [10]
    @param[in] n2 @dimensions [10]
*/
short nAddArrayShort(short *n1, short *n2);

/**IBMDAV* @function sumSquareMatrix
    @param[in] matrix @dimensions [N*N]
*/
double sumSquareMatrix( double* matrix, int N);

/**IBMDAV* @function reverseIntArray
```

```

        @param[inout] array @dimensions [size]
*/
int reverseIntArray( int* array, int size);

/**IBMDAV* @function funcInOut
        @param[inout] param1 @dimensions[n1*5]
*/
int funcInOut(double *param1, int n1);

/**IBMDAV* @function fAddArrayFloat
        @param[out] n1 @dimensions [1][10]
        @param[out] n2 @dimensions [1][10]
*/
float fAddArrayFloat(float **n1, float **n2);

/**IBMDAV* @function funcReturn
        @return @dimensions[*resultSize]
        @param[out] resultSize @dimensions[1][1]
*/
double* funcReturn(int **resultSize);

/***** 5) Function Tag :@return *****/
/**IBMDAV* @function reverseArrayIntb
        @param [inout] arrayInt @dimensions [10]
        @return @dimensions [10]
*/
int* reverseArrayIntb(int *arrayInt, int a);

/**IBMDAV* @function nAddArrayConstSignedInta
        @param n1 @dimensions [dim]
        @param n2 @dimensions [dim]
        @return @dimensions [dim]
*/
const signed int *nAddArrayConstSignedInta(const signed int n1[],
const signed int n2[], int dim);

/***** 6) Struct Tag *****/
/**IBMDAV* @struct myStruct */
typedef struct myStruct
{
        long lg;
        short sh;
        int x;
        float fl;
        double db;
        char ch;
        long long ll;
        long int li;
        short int shi;
        unsigned us;
}myStruct;

/**IBMDAV* @struct Details
        @param name @type string
*/
typedef struct structDetails
{
        char* name;
        int age;
        char sex;
}Details;

/**IBMDAV* @function copyStruct3

```

```

        @param[out] copyDetails @dimensions [1][10]
        @param[in] originalDetails @dimensions [10]
    */
void copyStruct3(Details** copyDetails, Details* originalDetails);

/***** 7) Void* *****/
/**IBMDAV* @function sum
    @param[in] array @dimensions [10] @type int */
public int sum(void* array, int size);

/**IBMDAV* @function nVariousVoidAdd
    @param arrayIn @dimensions [10] @type double
    @param[inout] arrayOut @dimensions [10] @type double
*/
int nVariousVoidAdd(void* arrayIn, void* arrayOut);

/***** 8) String *****/
/**IBMDAV* @function sum
    @param[in] str @type string */
public int length(char* str);

#if defined(__cplusplus)
};
#endif

```

Appendix C. Server Side IBM_DAV.conf settings

```
#
# Licensed Materials - Property of IBM
# Restricted Materials of IBM
# 10604 - IBM Dynamic Application Virtualization.
# Copyright IBM Corp. 2007, 2008 All Rights Reserved.
# US Government Users Restricted Rights
# - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with
#IBM Corp.
#

#####
#General configuration

#Log file location - change as desired
#ensure that the location specified has write access for the user running dav
dav.log.directory=/var/tmp/
dav.log.filename=ibm_dav.log

#####
#Broker configuration
#Applicable to the broker node only

#The broker machines address
dav.broker.listen.services.ip=127.0.0.1

#Port on which the broker will accept service notifications
#This entry must be available to all service compute nodes
dav.broker.listen.services.port=4321

#Port on which the broker will accept client requests
dav.broker.listen.clients.port=4320

#####
#Services configuration
#Applicable to compute nodes only

#Maximum number of requests that can be executed on a node at one time
dav.server.services.maxrequests=1

#The location of services on the system
dav.server.services.root=services
#Relative path specified resulting in path of "$IBM_DAV_PATH/services"
#Or if starting with "/" an absolute path i.e. =/usr/lib
#Will result in a path of "/usr/lib"
#If this entry is missing it defaults to "$IBM_DAV_PATH/bin"

#####
#Service Entries
#These entries will be automatically generated by the IBM DAV Virtualizer in a
#"changes_to_server_config_file.txt" file.
#The entries should be inserted here

#This is the sample library and its library path resulting in a location
#of "$IBM_DAV_PATH/services/Library"
dav.server.service.Library=Library get_Library_oai Library_oai
dav.server.service.Library.root=Library
dav.server.service.Library.listen.port=12300

#BLAS service
```

```
dav.server.service.blas=blas get_blas_oai blas_oai
dav.server.service.blas.root=blas
dav.server.service.blas.listen.port=12301

#LAPACK service
dav.server.service.lapack=lapack get_lapack_oai lapack_oai
dav.server.service.lapack.root=lapack
dav.server.service.lapack.listen.port=12302

#Service entry paths - additional library search paths

#dav.server.service.Library.root=/usr:Library
#This will produce a search path of "/usr:$IBM_DAV_PATH/services/Library"
#if the entry dav.server.service.root=services
#or a search path of "/usr:$IBM_DAV_PATH/bin/Library" otherwise
```

Appendix D. Client Side IBM_DAV.conf settings

```
#
# Licensed Materials - Property of IBM
# Restricted Materials of IBM
# 10604 - IBM Dynamic Application Virtualization.
# Copyright IBM Corp. 2007, 2008 All Rights Reserved.
# US Government Users Restricted Rights
# - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with
#IBM Corp.
#

#Which transport to use - DavTransport is the built-in transport
dav.transport.interface=DavTransport

#Log file location - change as desired
dav.log.directory=c:\
dav.log.filename=ibm_dav.log

#Connections to servers
dav.servers=myserver
myserver.dav.server.ip=127.0.0.1
myserver.dav.server.port=4320

# client service timeout
# if a sequence of client calls to the same service e.g davrand ( initialize,
# perform, terminate ) have large intervals between the calls
# the timeout for the service should be increased according to the client
# requirements (the default is 1000 milliseconds) e.g.
# dav.server.service.davrand.timeout=2000
```

Appendix E. IBM DAV Executable Options

To start all services:

```
$IBM_DAV_PATH/bin/dav start
```

To stop all services

```
$IBM_DAV_PATH/bin/dav stop
```

To see which services which are currently running

```
$IBM_DAV_PATH/bin/dav status
```

To start a service when other services are already running – run dav start again

```
$IBM_DAV_PATH/bin/dav start
```

The first time dav start is run it loads all service information from the config file so rerunning dav start will start non-running services.

A single service can be specified even if other services are already running

```
$IBM_DAV_PATH/bin/dav start -t IBM_DAV_PATH/bin/dav start -t <service name>  
[-p <port>]
```

Note:

- this has the same effect as running dav start unless a different port is specified
- the port a service is to run on should be specified in the server-side IBM_DAV.conf. Thus there should be no need to specify the port on the command line.

Appendix F. makefile for IBM DAV Virtualizing a Linux application

```
LIBNAME=Library

All: $(LIBNAME)Driver

$(LIBNAME)Driver: $(LIBNAME)Driver.cpp
g++ -g -o $@ $^ -L. -L$(IBM_DAV_PATH)/lib -l$(LIBNAME)_oai -law
-liaiConnect_oai -lDavTransport -lm

clean:
$(RM) *.o $(LIBNAME)Driver
```

Appendix G. Troubleshooting Errors

This section gives information on how to Troubleshoot Errors

Semantics and DAVGEN

DAVGEN reports compilation errors:

- DAVGEN only supportst c header files (not c++)
- Check semantics applied to the c header file
 - Check syntax, check semantics using Semantic Verifier if available.
- Ensure that you aren't trying to utilize functionality not supported by IBM DAV.

Note: Only two-dimensional arrays are supported, no higher dimensions.

Note: Only one level of #include is currently supported.

IBM DAV Deployment

Deployment failed:

- Check connectionDetails.txt for incorrect login details.
- Ensure that the correct path on the server was given to the script for the header and library.

Start IBM DAV Service on Server

Some services failed to initialize:

- Check that you have all the necessary rpms installed, and that they are of the correct version.

Running your application

10061 – No connection could be made because the target machine actively refused it:

- Check that the DavServiceBroker and service have been started
- Check that the server details listed in the Clients IBM_DAV.conf are correct.
- Ensure that a local firewall isn't blocking access to the client machine

Index

A

Automatic Deployment 33

B

BLAS 24

C

Comment Format 15
Configuring
 Virtualizer for Visual C++ 11

D

dav status 36
Dimension definitions 20

E

Excel application 44
Excel Client Application 41

F

FFT 26
Function tag 16
Function tag: @dimensions 18
Function tag: @param 16
Function tag: @return 18

G

GCC 4.1.2 12
Generating DAV Libraries 23

I

IBM_DAV_PATH 12
IBM_DAV.conf 35
in 16
inout 16
Installation
 Client-side developer Linux 7
 Client-side developer Windows 6
 Client-side Linux 6
 Client-side Windows 6
 Server-side Linux 7
Installation Components 6
 Client 6
 Server 6
 Virtualizer 6
Installation instructions 5
Installing
 MS Platform SDK (R2) 10
 Visual C++ 2005 Express 9
 Visual C++ 2005 Redistributable 10

J

Java application 44
Java Client Application 42

L

LAPACK 25
Launch the Service 35
LD_LIBRARY_PATH 12
Library Semantics 28
Library tag 15

M

Monte Carlo Library 27

O

Operating Systems
 Client 3
 Server 3
out 16
Overview 1

P

Platform Requirements 3

R

Running a Service
 Dedicated Mode 36
Running the client application 43

S

Sample Header File 51
Sample Libraries 24
Semantic Verification 29
Semantics 15, 16, 18, 19, 20
 Sample Header File 53
Server Side Deployment 33
Sobol 27
Software Requirements
 Linux 3
 Windows 3
Start the Service Broker 35
Stopping DAV Service 36
Strings 20
Struct tag 18

T

Tag Types 15
Troubleshooting Errors 65

V

Virtualizer Configuration 9
 Visual C++ 2005 Express 9
Virtualizing the client application 39
 Linux 39
 Windows 39
Visual C++ 2003 12
Void* 19

W

What is IBM DAV 1

X

XLL Add-ins 47
XLL Assumptions 50
XLL Limitations 49
XLL Use 47



Printed in USA