Co>Operating System Administrator's Guide

(with Addendum)

July 2003 > Part Number AB0386

Ab Initio Software Corporation
201 Spring Street > Lexington, MA 02421 > Voice 781.301.2000 > Fax 781.301.2001 > support@abinitio.com

Contents

Preface

Purpose

This manual provides user-level information about the Ab Initio Co>Operating System. It also covers selected topics for system administrators.

Audience

This manual is intended for users who work with Ab Initio applications but who do not necessarily create them. Such a user may need to:

- Create or manage parallel file systems.
- Manage collections of metadata (which describe the format of datasets).
- Manage the execution environment (the set of servers to be used and the configuration of the executing shell).
- Monitor the progress of application runs.
- Deal with system failures.

Developers of Ab Initio applications also need this information. In addition, developers should read the documentation for their preferred environment: *Ab Initio Shell Development Environment User's Guide* or the Ab Initio Graphical Development Environment on-line Help) and the reference manuals for other development facilities: *Data Manipulation Language Reference*, *Component Reference*, and *Database Reference*.

Documentation conventions

Unless otherwise noted in the text, we use the following documentation conventions:

literal names	Bold represents literal words, keywords, characters, or values, except for configuration, environment, and system variables, which are not bold.
AB_HOME	Words whose letters are all in uppercase are configuration, environment, or system variables.
user-supplied values	Italic words or characters represent variables or values that you must supply.

`source code`	A fixed-width font represents command-line sessions, code fragments, and examples.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
a | **b** | **c**	The logical **or** symbol separates alternatives.
{ }	Curly braces surround a series of choices from which you must use one, except as indicated when they are part of the syntax.
[ ]	Square brackets surround optional elements, except as indicated when they are part of the syntax.
⇒	An arrow indicates one of the following: • The result of computations. For example: **string_concat("abcd", "efgh")** ⇒ **"abcdefgh"** • A command or line of code continued on the next line
`sample user input`	In interactive command-line examples, information that you must enter appears in bold while computer output appears in normal weight.

1>

Overview

The Ab Initio Co>Operating System unites a network of computing resources - CPUs, storage disks, programs, datasets–into a production-quality data-processing system with scalable performance and mainframe-class reliability.

The Co>Operating System is layered on top of the native operating system(s) of a collection of servers. It provides a distributed model for process execution, file management, process monitoring, checkpointing, and debugging. A user may perform all these functions, on any or all servers, from a single point of control.

This chapter presents an overview of the Co>Operating System in action, using a sample parallelized business application. The remainder of this volume describes Co>Operating System services and utilities.

User control

The Co>Operating System provides a number of services that are controllable by the user. These may take the form of:

- **mp** *name* commands - used in building applications as shell scripts
- **m_***name* commands - used to invoke shell-level utilities
- Environment variables - used to configure the executing shell

mp commands

The **mp** refers to the Ab Initio command interpreter, and **mp** commands take the form:

> **mp** *command-name argument1 argument2* ...

The **mp** commands are typically used within an application script to specify program components and to indicate the flow of data. A script may also include **mp** commands that set checkpoints, invoke the application, clean up temporary files, and so on. Ab Initio provides the Shell Command Interface as the environment for building component-based applications with mp commands. These are described in the *Ab Initio Shell Development Environment User's Guide*.

Once an application is production-ready, the user who executes it rarely needs to use mp commands directly. However, it may be necessary to look inside an application script to check the location of checkpoints, the provisions for monitoring, the impact on environment variables, etc. For this reason, this chapter provides a brief introduction to Ab Initio component-based applications.

m_*name* **commands**

Commands with the prefix **m_**, such as **m_cp** or **m_dump** or **m_rollback**, are Co>Operating System shell-level utilities. These utilities are typically used for managing parallel files, managing metadata, recovering a checkpointed process, and so on. This manual focuses on these **m_** commands.

The Co>Operating System provides the commands that generate or manage Ab Initio metadata (described in Chapter 3> "Metadata management" on page 31). For information on metadata itself, see the *Ab Initio Data Manipulation Language Reference*.

Environment variables

The Ab Initio system makes extensive use of environment variables. Some may be set centrally by the system administrator; others may be set locally for a user's environment or for a particular application.

This manual describes the Ab Initio environment variables.

When running applications, please note that environment variables are passed downward only. When a script is invoked, its environment variable settings will initially be those of the shell that invoked it. If the script changes any of the values, those values are changed only for the course of the application.

Before you begin

Make sure these environment variables are set before you attempt to use any Ab Initio shell command or other feature:

- AB_HOME must be set to the root directory of the installed Ab Initio software, which is typically **/usr/local/abinitio**. For example, in Korn shell syntax:

```
export AB_HOME=/usr/local/abinitio
```

- PATH must be set to search the bin directory under the root of the installed Ab Initio software:

```
export PATH=$AB_HOME/bin:$PATH
```

These settings enable your shell to locate the installed Ab Initio software.

Going parallel: A sample application

A parallel system can be any collection of servers, PCs, clusters, SMPs, and/or MPPs that have the Ab Initio software installed.

An application becomes parallel when some or all of its components - datasets and processing modules - are replicated into a number of partitions. That is:

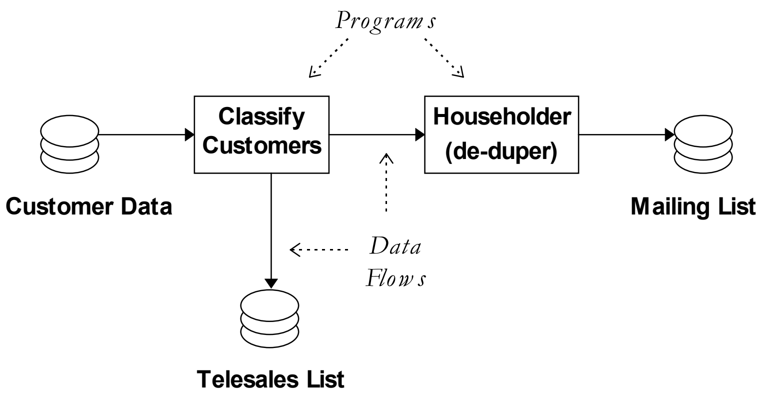
- A parallel file is a file consisting of an ordered set of individual files, usually located on different disks or on different systems. Ab Initio calls these parallel files “multifiles.”
- A parallel program is one that runs n times from the same executable, resulting in n processes, which are usually on different CPUs. Such a system can process multiple partitions of the data at the same time.

This section provides a brief description of an Ab Initio component-based parallel application.

Users who plan to develop applications like this need the more detailed information given in the *Ab Initio Shell Development Environment User’s Guide*. This discussion is intended for users who will be executing parallel applications and would like to have a general description of their behavior.

An application

Suppose that an enterprise wants to classify its customers as candidates for either telemarketing or direct mail. The processing steps needed are these:



- A scoring program classifies each customer record as a target for telesales or direct mail.
- Records for customers targeted for telesales are sent to a file.
- Records for customers targeted for direct mail are sent through another program–a "de-duper" that removes duplicate addresses–and then placed in a second file.

Forms of parallelism

The forms of parallelism are pipeline parallelism and data parallelism.

Pipeline parallelism

The Ab Initio system allows the two processing stages - classification and deduping - to run at the same time on different processors. Performance improvements result from this *pipelined* processing of the data.

A limitation of pipeline parallelism is that it can be extended only to as many processors as there are processing stages.

Data parallelism

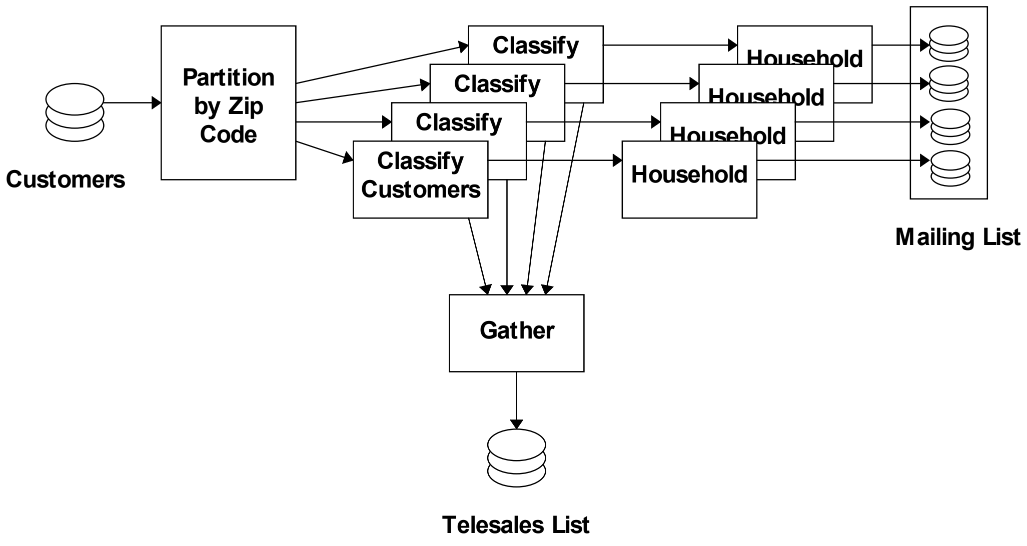
Another way of getting parallel performance is to divide the data into an arbitrary number of partitions, with each on a separate disk. Then we can use a collection of processors, each one accessing one of the disks and processing the data it holds.

With data parallelism like this, we should achieve speedups that are nearly linear with the number of processors employed.

Parallelizing the application

If the input data, customer records in this case, is stored serially, we need to build a partitioning component into the program. On the output side, let's assume that the telesales targets are to be gathered into a single file, while the mailing targets will be stored in parallel files (that is, in an Ab Initio multifile) to facilitate later processing by other parallel applications.

The necessary tasks are performed by this application, where each disk represents a file (or a partition of a multifile) and each box represents a program or a partition of a program:



The customer data is grouped by zip code into four partitions.

Each partition is processed independently, classifying each of its customer records as a target for telesales or direct mail.

Records for telesales candidates are gathered into a single file.

Records for direct mail candidates are sent through the **Household** (de-duper) program and finally into a multifile.

Ab Initio features used

This application uses multifiles, components, layouts, and data transport.

Multifiles

In this application, the mailing list data remains partitioned across four files. The Co>Operating System treats the four as a single manageable entity–a four-partition multifile.

Multifiles give users the convenience of central control of distributed data files, and they provide the scalability and the kinds of access patterns that parallel applications require. For these reasons, multifiles are a better choice for distributing data than network file systems such as NFS, SMB, or CIFS. Ab Initio multifiles work together seamlessly with Ab Initio parallel applications.

Program components

This application combines four program components: the Partitioner, the Customer Classifier, the Household, and the Gatherer. These programs probably came from three different sources:

- The Partitioner and the Gatherer are taken without modification from the Ab Initio Components Library.
- The Customer Classifier is likely to be a user-written code that applies business logic specific to the enterprise. The Ab Initio system provides facilities to encapsulate such codes as application components.
- The Household "de-duper" might come from any of a number of commercial vendors of data-processing programs. Ab Initio software can integrate such programs, without modification, into the application.

Layouts: Where things happen

A layout specifies how some part of an application is partitioned. That is, a component's layout specifies the number and location of its partitions, giving a hostname and pathname for each partition. Every component of an application has a layout (even if it has only one partition).

Layouts are specified in the shell script that implements the application. Usually, a user who executes the application need only check that the specified resources are available.

Data transport

Transparently to the user, the Co>Operating System manages any necessary movement of records from one place (*hostname*/*pathname*) to another. The system automatically selects among various transport mechanisms available through the underlying operating system to optimize data movement. These may include TCP/IP sockets, special communication hardware, FIFOs, shared memory regions, etc.

If an application is written to use a specific transport mechanism, such as named pipes, the Co>Operating System will automatically insert transport adapters to convert the flow of records to that particular mechanism.

Inside the application

The application described in the preceding section can be constructed easily from the **mp** commands provided in the Ab Initio Shell Environment.

In this environment, a developer writes a shell script that identifies the datasets and their layouts, the processing steps and their layouts, and the flows of data from one stage to the next. The script also identifies the metadata, that is, the format of the data in each of the flows, and any special actions such as checkpointing, testing/debugging, etc.

This section provides a brief look inside an Ab Initio application.

Ab Initio shell commands

The Ab Initio **mp** commands look like this:

mp *command argument-1 argument-2* ...

Each line begins with the word **mp**, which is the name of the command interpreter that interprets the remainder of the line. Although the **mp** commands are simply shell commands, they are most often used in scripts rather than interactively.

Some of the **mp** commands that you will see in every application are in the following Table .

Commonly used **mp** commands

Command	**Use**
mp job	Establish the "framework"
mp ifile, **mp ofile**, ...	Define data file components
mp metadata	Define metadata objects that describe data in flows
mp hash-partition, **mp gather**, ...	Define program components
mp straight-flow, **mp fan-out flow**, ...	Describe the flows of data
mp run	Run the application

An application script

Shown below is the entire script (**classify-customers.mp**) needed to implement the customer-classification program we have been examining:

```
#! /bin/ksh
set -e

mp job classify-customers


#  Component       Name            Arguments
#  ---------       ------          ---------
mp ifile  customers       file:customers
mp ofile  telesales     file:telesales
mp ofile  mailing-list    mfile:/mfs/mlist

mp hash-partition partition     "zipcode" -layout customers
mp classifier classify       -layout mailing-list
mp householder household     -layout mailing-list
mp gather gather     -layout telesales


#  Define record format
mp metadata       format         -file customers.dml


#  Flow type    Name From Port      To Port          Data format
#  ---------    ---- ---- ----      -------          -----------
mp straight-flow f1  customers.read partition.in     -metadata format
mp fan-out-flow  f2  partition.out  classify.input   -metadata format
mp straight-flow f3  classify.mail  household.in     -metadata format
mp straight-flow f4  household.out  mailing-list.write -metadata format
mp straight-flow f5  classify.phone gather.input     -metadata format
mp fan-in-flow   f6  gather.output  telesales.write -metadata format


# Run the Application
mp run
```

This script specifies the application's data and processing components, the metadata, the data flows, and the execution command. Later examples show how monitoring, checkpointing, and so on, appear when they are built into a script.

2>

File Management

The Ab Initio Co>Operating System provides facilities for managing large datasets that span many disks or are distributed across multiple servers.

The Co>Operating System supports multifile systems, which allow users to manage a set of distributed files as a single entity. When your data is stored in Ab Initio multifiles, you can apply familiar file operations to the whole collection of files from a central point of administration. Further, an Ab Initio application can access files anywhere within an enterprise, using the familiar URL (Universal Resource Locator) syntax of the World Wide Web.

This chapter describes:

- Data parallelism and the layouts of multifiles and programs
- The syntax for referencing files locally and remotely
- Facilities for creating and managing multifile systems
- Facilities for creating multifiles or for substituting other multifiles at run time

Data parallelism and layouts

The components of an Ab Initio application–programs and files–can easily be made parallel. A parallel component exists as a number of partitions which replicate the structure of a non-parallel component, which we shall call a "standard" or "serial" component. That is:

- A parallel file consists of an ordered set of standard files, usually on different disks or on different systems.
- A parallel program is one that runs *n* times from copies of the same executable, resulting in *n* processes, which are usually on different CPUs.

The number and locations of the partitions of a component are described to an Ab Initio application as a layout. You specify a layout when you do either of these two things:

- Create a multifile system, a place where parallel files (multifiles) are stored.
- Construct a layout object, used to describe the parallelism of a component program in a graph application. (Layout objects are described in the *Ab Initio Shell Development Environment User's Guide*.)

A layout is a list of *hostname/pathname* pairs. Each entry in the list represents one partition of the multifile or program component. Very commonly, the layouts used in an application are derived from the layouts of parallel files.

NOTE: Besides data files, the Ab Initio system also supports SAS datasets and many commonly used RDBMSs. These datasets are accessed by means of special interface components in the graph application. As with data in files, database or SAS data that is stored in parallel may be processed in parallel by the hosts on which it resides. Alternatively, data stored serially may be "fanned out" to multiple servers for processing.

Multifile systems

In the Ab Initio system, a parallel file is called a multifile.

Multifiles

Multifiles are parallel files composed of individual files located (usually, but not necessarily) on different disks or on different systems. The individual files are referred to as the partitions of the multifile.

Visualize a standard directory tree containing subdirectories and files. Now imagine *n* identical copies of the tree located on several disks. These are the partitions of the multifile system. To complete the picture, we must add one more exact replication of the tree, which is called the control partition. The control partition is located on some host computer. A diagram of a multifile system appears on page 17.

When taken as a whole, the individual partition files constitute a single virtual file. The contents of this virtual file can be read or written at the sum of the individual disk bandwidths.

Multidirectories

Multifiles are stored in parallel directories called multidirectories, which reside in a multifile system. An Ab Initio multifile system is a replication of the native operating system's directory tree structure, designed such that each partition can hold a subset of data to be processed. The multifile system has an additional partition that contains control information.

Control files

A multidirectory consists of a control directory (a directory somewhere in the control partition), and the *n* corresponding directories from the partitions of the multifile system. Similarly, a multifile consists of a control file (a file somewhere in the control partition), and the *n* corresponding files from the partitions of the multifile system.

There is no user data in the control file. The individual partition files of the multifile contain the user data.

Referencing multifiles

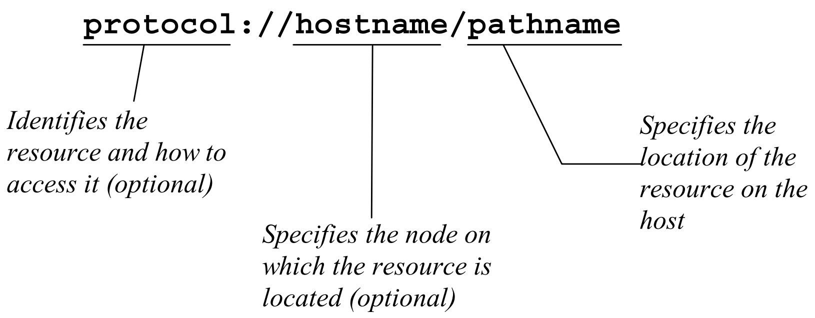
You reference a multifile system by referencing the root of the control partition. You reference multifiles and multidirectories by referencing the files and directories within the control partition.

In graph programming, it is possible - in fact common - to specify a layout for parallel program execution simply by referencing a multifile (or multidirectory or multifile system).

URLs

The Ab Initio system uses Universal Resource Locators (URLs) to identify distributed resources. Each partition of a layout - that is, each *hostname/pathname* pair–is specified by a URL, as used on the World Wide Web.

A URL is constructed like this:


protocol://hostname/pathname
Identifies the resource and how to access it (optional)
Specifies the node on which the resource is located (optional)
Specifies the location of the resource on the host

In Ab Initio directory and file references:

- The *protocol* field indicates whether the file is a standard (serial) file or a partition of a multifile.
- The *hostname* field specifies the computer where the partition resides.
- The *pathname* is a standard pathname, in the form accepted by the host’s native OS, indicating the partition’s location on the host.

Protocol field

The optional protocol field of a URL may contain a file type:

- **mfile:** for multifiles
- **file:** for standard (serial) files

This item is optional since the Ab Initio system can *usually* determine by examination whether a file is a multifile or a standard file. However, it is good practice to use these prefixes to prevent ambiguity.

Hostname field

Hostnames in a URL are Internet host names. For example, a computer known as **yang.com** on the Internet or on a disconnected intranet would be referred to by that name in Ab Initio URLs.

Computer name localhost

The special Internet name **localhost** refers to the system on which the referencing program is running. However, it is unwise to create multifile systems with **localhost** as the host name of the control partition or any of the data partitions, since its meaning changes depending on the computer from which it is referenced. See the description of the file utility **m_mkfs** below.

Hostnames on IBM SP2 system

The hosts comprising an IBM SP-2 are interconnected with two networks, a standard (slower) Ethernet LAN and the high-speed switch. As a result, each host of an SP-2 has two Internet hostnames, one for accessing the it via the Ethernet and the other for accessing it via the high-speed switch.

By default, AIX uses the Ethernet hostname when communication channels are opened. To ensure that the Ab Initio system sends data across the switch rather than the Ethernet, you must take the following steps:

1. First, when creating a multifile system, use the switch hostname in the URLs that define the partitions. For example, if **host1** and **host2** are the Ethernet hostnames for two computers, and **host1s** and **host2s** are their switch hostname counterparts, then you should create multifile systems using this command line:

   ```
   m_mkfs mfs1 file://host1s/data/mfs1a \
       file://host2s/data/mfs1b
   ```

2. Second, it is desirable to ensure that the Ab Initio system will use the switch hostname for communication between the host where an application runs and the remote processing hosts. From the host where the **mp** commands will be issued, set the environment variable AB_HOST_INTERFACE to the switch hostname of that host. For example, if the **mp** commands will be executed from a

script on host **host20** and its switch hostname is **host20s**, you should set:

```
export AB_HOST_INTERFACE=host20s     [ksh]

setenv AB_HOST_INTERFACE host20s     [csh]
```

Referencing standard files

Wherever the Ab Initio system accepts the name of a file, you can indicate either a multifile or a standard (non-partitioned) file. The system lets you refer to standard files using URLs and so they can be located anywhere on the network (as long as the referenced host is running the Ab Initio system). The URL protocol **file:** is used to signify standard files. For example:

```
file://host2/u/george/input.dat
```

Segmented files as partitions

The partitions of a multifile may be standard files, or they may be segmented files (also called vfiles, for "vector files").

File segmentation is transparent to the user. A segmented file is a logical file composed of an ordered set of standard files, which all reside in a single directory on the same host. That directory is called the vfile's control directory. A reference to the vfile as a logical unit is actually a reference to the URL of the vfile's control directory.

You use Ab Initio file utilities (described on page 19) to create and manage vfiles, and you use **mp** file commands (**ifile**, **ofile**, and **iofile**) to access vfiles from graph programs.

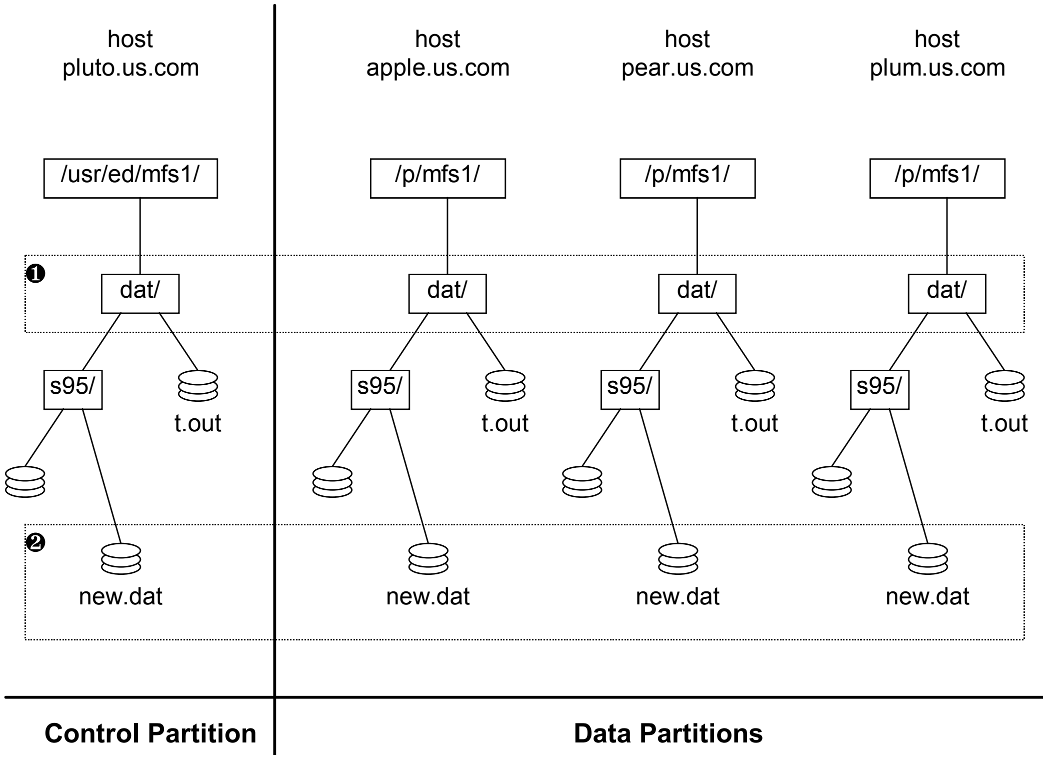
Segmented files have an important advantage over standard files. They can be arbitrarily large, since they are not constrained by OS file-size limits.

File segmentation is orthogonal to file partitioning, although Ab Initio enforces the restriction that only multifiles can be segmented. Each partition of a multifile may have multiple segments, and the number of segments may be different on different hosts. An attribute of a vfile is segment size, which is a maximum. Depending on the amount of data stored on the respective hosts, a partition may have fewer segments or smaller segments than other partitions of the same multifile.

A sample multifile system

The diagram below shows a multifile system with its control partition at **//pluto.us.com/usr/ed/mfs1**. This multifile system has three partitions and, consequently, so do all the multidirectories and multifiles it contains. The three partitions of the multifile system are at various locations on the disks of three computers: **apple.us.com**, **pear.us.com**, and **plum.us.com**.

The following is a three-partition multifile system:



The root of the control partition is used to name the multifile system. Therefore, this multifile system has the following name:

```
mfile://pluto.us.com/usr/ed/mfs1.
```

Highlighted in dotted lines are two components of this multifile system:

1. A multidirectory: **mfile://pluto.us.com/usr/ed/mfs1/dat**
2. A multifile: **mfile://pluto.us.com/usr/ed/mfs1/dat/s95/new.dat**

File management utilities

The Ab Initio file utilities provide the ability to manipulate files and directories, including multifiles and multidirectories, regardless of where in the enterprise they are located.

For the most part, the Ab Initio utilities work like, and are named like, their UNIX counterparts with the prefix **m_** added. For example, the **m_cp** command works like the UNIX **cp** command.

The important difference is that with the Ab Initio file utilities, you use URLs to specify file and directory paths. This allows files to be located on remote hosts and to be multifiles or multidirectories. Some other small semantic differences exist, usually related to error reporting or usage of file protections, and these will be mentioned where appropriate.

m_mkfs [*options*] *control_url partition_url* [*partition_url* ...]

m_mkfs [*options*] *control_url* **-**

m_mkfs [*options*] *control_url* -f *file_with_pathnames*

Create a multifile system. The *control_url* is a URL specifying the directory to be created; it will be the root of the control partition of the multifile system.

- The prefix of the *control_url* argument must be **mfile:** or may be omitted, in which case **mfile:** is assumed.
- The hostname portion of *control_url* may specify any accessible remote host.

In the first form, the data partitions of the multifile system are specified in the list of URLs. In the second and third forms, the partition URLs are read from the standard input or from the file *file_with_pathnames*, respectively, separated by whitespace or newline. The partition URLs must bear the **file:** prefix, or if omitted, **file:** will be assumed.

Invoking **m_mkfs** with no arguments generates a message summarizing its usage.

It is best to specify explicit host names for all partitions. It is unwise to create multifile systems with no host name, or with a host name of **localhost**, for the control partition or any data partition, since its meaning changes depending on the host from which it is referenced.

Also, it is best to avoid network-mounted file systems: if a partition's file system is mounted on more than one host, specify the host its disk is physically connected to.

Options:

The following are the **m_mkfs** options:

-m *mode* or **-mode** *mode*	Set the protection for the created multifile system. The *mode* option is one of the symbols accepted by the UNIX **chmod** command.
-mvfile	Specify that all multifiles in a multidirectory (and in all its multi-subdirectories) will have vfiles (segmented files) as their partitions. (By default, the multifiles in a multidirectory have standard files as their partitions.)
-max *bytes* or **-max-segment-size** *bytes*	Specify the maximum segment size for vfile partitions. (The default is a system-specific value.) Note that the value supplied serves only as an approximation of the maximum segment size. It is advisable to use a value somewhat smaller than the actual desired value. This option is only meaningful when **-mvfile** is also specified.

Example

Create the multifile system shown on page 17:

```
m_mkfs    //pluto.us.com/usr/ed/mfs1 \
          //apple.us.com/p/mfs1 \
          //pear.us.com/p/mfs1 \
          //plum.us.com/p/mfs1
```

Its layout (the locations of its partitions) is:

```
//apple.us.com/p/mfs1
//pear.us.com/p/mfs1
//plum.us.com/p/mfs1
```

m_rmfs *path*

Delete a multifile system.

The *path* must be of type **mfile:** (if the prefix is omitted, **mfile:** is assumed). The *path* must be the current root of a multifile system that is empty of multifiles and multidirectories.

m_mkdir [**-m**[**ode**] *mode*] [**-mvfile**] [**-max-segment-size** *bytes*] *path*

Create a multidirectory.

The *path* is the directory under which the directory will be created. The *path* must be of type **mfile:** (if the prefix is omitted, **mfile:** is assumed). The *path* must be within a multifile system, but it must not refer to an multidirectory that currently exists.

The *mode* sets the protection for the created directory, specified as one of the symbols accepted by the UNIX **chmod** command.

The **-mvfile** specifies that all multifiles in the multidirectory and in all its multi-subdirectories will have vfiles as their partitions. (By default, the multifiles in a multidirectory have standard files as their partitions.) The size of the vfile segments can either default to some system-specific value, or it may be specified with the **-max-segment-size** option. Note that the value supplied serves only as an approximation of the maximum segment size. It is advisable to use a value somewhat smaller than the actual desired value.

Examples

Create the **/dat** multidirectory in the example above:

```
m_mkdir mfile://pluto.us.com/usr/ed/mfs1/dat
```

If you were to create a multifile named **donut** in this multidirectory, it would be referenced by this URL:

```
mfile://pluto.us.com/p/mfs1/dat/donut
```

Its layout (the locations of its partitions) would be:

```
//apple.us.com/p/mfs1/dat/donut
//pear.us.com/p/mfs1/dat/donut
//plum.us.com/p/mfs1/dat/donut
```

m_rmdir *url* [*url* ...]

Remove empty multi-directories or directories.

m_rm [**-f**] [**-I**] [**-r**] [**-R**] [**-v**] *url* [*url* ...]

Remove multifiles or files.

The following are the **m_rm** options:

-f	Force, as in the UNIX **rm** command.
-I	Interactive, as in the UNIX **rm** command.

-r	Recursive, as in the UNIX **rm** command.
-R	Recursive, as in the UNIX **rm** command (same as **-r**).
-v	Verbose. Emit the URL for each file, multifile, directory, or multidirectory removed.

m_chmod *mode url* [*url* ...]

Change the protection of a file or directory of any type.

Setting protection on multifiles entails setting the identical protection on the control file and on each partition file. Protection levels are specified as one of the symbols accepted by the UNIX **chmod** command.

m_touch *url* [*url* ...]

If the file specified by *path* exists, update its access time to the current time. If the file does not exist, create it as an empty file.

Path may be of type **file:** (a standard file is created), or **mfile:**. In the latter case, *path* must be within an existing multifile system.

m_cp [**-report**] **-a**[**ll-at-once**] **-v**[**erbose**] *source_file_url_1 source_file_url_2 ... dest_url*

Make a copy of each *source_file* with the name and path specified by *dest_url*.

If *dest_url* is a directory, the files are copied to that directory under their original names. If exactly one source file is given, *dest_url* may optionally bear a new name for the file. Each *source_file_url* must be a file, not a directory. Both *source_file_url* and *dest_url* must either be standard files (type **file:**) or multifiles (type **mfile:**) located within the same multifile system.

The following are the **m_cp** options:

-a or **-all-at-once**	Copy all source files at the same time. This may complete more quickly, but it may also consume more system resources. The default is to copy source files one at a time.
-report	Produce monitor reports as specified in the environment variable AB_REPORT. See "AB_REPORT settings" on page 58.
-v or **-verbose**	Emit a message to the standard output as each file is copied.

m_mv *source_path1 source_path2 ... dest_path*

Move files or directories specified by *source_path* to the name and path specified by *dest_path*.

If *dest_path* is a directory, the files are moved to that directory under their original names and deleted from their original locations. If exactly one *source_path* is given, *dest_path* may optionally bear a new name for the file. The source and destination paths must either be standard file/directory paths (type **file:**) or multifiles/multidirectories (type **mfile:**).

m_ls [**-dRlarsctum**] [**-partitions**] [**-max-skew** *skew*] [**-no-commas**] *path path ...*

List the specified files or directories. The argument(s) *path*... may be a file or directory of type **mfile:**, or a file or directory of type **file:** (a standard file or directory). The optional flags **-dRlarsctu** are interpreted as with the UNIX **ls** command. For information on the **-m** flag, see "Machine-readable output" below.

The following are the **m_ls** options:

-partitions	Following the line for each *path*, print a line for all its partitions. If *path* is not an multifile or multidirectory, this option is ignored. The line corresponding to a partition is distinguished by a plus sign (+) in the column to the right of the permissions. This option is meaningful only when **-l** is supplied.
-max-skew *skew*	Following the line for each *path*, print a line for all its partitions with skew greater than *skew*. The argument is a percentage supplied as a decimal integer, e.g., **35**. (Skew is an indication of uneven partition size. See page 28 for an explanation.) This option is meaningful only when **-l** is supplied.
-no-commas	Print directory sizes without commas. (Default is printing with commas for legibility.) This option is meaningful only when **-l** is supplied.
-l	Long format. This format is similar to the long format printed by the UNIX **ls** command, but it has two extra columns.

The following is an example of the long format (**-l**).

The command:

```
m_ls -l mfs
```

```
drwxrwxr-x D   owner  group          512 May 29 17:39      dir
-rw-rw-r-- M   owner  group   21,400,932 May 29 17:02   5% out.dat
```

The **D** and **M** indicate that the entry is a multidirectory and multifile, respectively.

The **5%** is the file's skew, computed on the number of bytes in each partition.

Both these columns are left blank for serial files and serial directories.

Machine-readable output

The **-m** flag specifies machine readable output. It changes the format of the output of **m_ls -l** argument so that every line of output has the same number of fields.

The **-m** argument affects the second and next to the last column in the output as follows:

In the second column:

- With just the **-l** argument, multidirectories are marked **D** and multifiles are marked **M**, but serial files and directories are blank.
- With **-lm**, serial directories are marked **d** and serial files **f**.

In the next to last column

- With just **-l**, the skew of multifiles is shown in the next-to-last column.
- With **-lm**, the corresponding position is marked with a hyphen (**-**) for items to which skew does not apply.

Consider the following example. Multifile directory **x** contains a serial file, a serial directory, a multifile, and a multidirectory.

The command **ls -l** produces the following output:

```
$ m_ls -l x
drwxrwxr-x D   ephraim  staff          192 Dec 20 19:31        multi-dir
-rw-rw-r-- M   ephraim  staff            36 Dec 20 19:31    38% multi-file
drwxrwxr-x     ephraim  staff            96 Dec 20 19:30        serial-dir
-rw-rw-r--     ephraim  staff             0 Dec 20 19:30        serial-file
```

Notice how the addition of the **-m** option in the **ls -lm** command makes the number of columns consistent in the following output:

```
$ m_ls -lm x
drwxrwxr-x D   ephraim  staff          192 Dec 20 19:31       -  multi-dir
-rw-rw-r-- M   ephraim  staff           36 Dec 20 19:31     38% multi-file
drwxrwxr-x d   ephraim  staff           96 Dec 20 19:30       -  serial-dir
-rw-rw-r-- f   ephraim  staff            0 Dec 20 19:30       -  serial-file
$
```

m_du [**-s**] [**-partitions**] [**-max-skew** *skew*] [**-no-commas**] *url* [*url* ...]

Print to **stdout** the disk usage for each item specified by a *url* in units of 1024-byte blocks; the item's skew, computed on the number of blocks occupied by each partition; and the URL for the item. This utility supports multifile systems, multidirectories, and multifiles, as well as ordinary (standard) directories and files. The skew column is left blank for directories and for serial files.

The following are the **m_du** options:

-s	Following the line for each *url*, print the total for the specified URLs, without the totals for all the subdirectories of the URLs.
-partitions	Following the line for each *url*, print the size for all its partitions. If a URL is not an multifile or multidirectory, this option is ignored. The line corresponding to a partition is distinguished by a plus sign (+) in the column to the right of the permissions.
-max-skew *skew*	Following the line for each *url*, print the size in bytes for all partitions with skew greater than *skew*. The argument is a percentage supplied as a decimal integer, e.g., **35**. (See page 28 for information on file skew.)
-no-commas	Print block counts without commas. (Default is printing with commas for legibility.)

m_df [**-partitions**] [**-max-skew** *skew*] [**-no-commas**] [**-terse**] [**-fullest**] *path* [*path* ...]

For each *path*, print to **stdout** the following information about the containing multifile system or (standard) file system:

1024-blocks	The amount of disk space, in 1024-byte blocks, occupied by the file system in which the partitions or file reside.
Used	The amount of disk space occupied by all multifiles or files in the file system.
Avail	Available space, indicating the largest multifile that can be created assuming that its partitions are of equal size.
Cap	Capacity: Used divided by 1024-blocks, expressed as a percentage.
Skew	The skew of the multifile system, computed on Avail. This column is left blank for standard file systems.
Filesystem	The URL of the multifile system or file system.

The following are the **m_df** options:Options:

-partitions	Following the line for each URL, print the size for all its partitions. If a path is not an multifile or multidirectory, this option is ignored. The line corresponding to a partition begins with a plus sign (+).
-max-skew *skew*	Following the line for each URL, print the size in bytes for all partitions with skew greater than *skew*. The argument is a percentage supplied as a decimal integer, e.g., **35**. (See page 28 for information about file skew.)
-no-commas	Print directory sizes without commas. (Default is printing with commas for legibility.)
-terse	Limit the output to a single number per line. That number is the number of blocks available in the specified file system. Use of **-terse** implies **-no-commas**, since its purpose is to make this information easily usable in scripts.
-fullest	Print information about the partitions of the multifile system that have the fewest disk blocks available.

m_expand [**-n** | **-hosts** | **-paths** | **-mfs**] [**-native** | **-unalias** *mode*] *url*

Prints to **stdout** various information about the multifile, multidirectory, file, or directory. If *url* does not exist, the result is determined by *url*'s parent directory. The *url* is taken to be a standard file if the parent directory is a standard directory. The *url* is taken to be a multifile if the parent directory is a multidirectory.

Either zero or one option must be specified.

-n	Print the number of partitions (**1** if the file is not a multifile)
-hosts	Print the hostnames of the partitions.
-paths	(The default.) Print the pathnames of the partitions.
-mfs	Prints the pathname of the multifilesystem.
-native	Causes **m_expand** to display native pathnames rather than URLs in the expansion.
-unalias	Controls resolution of host aliases during expansion. Following is the syntax of **-unalias**: **m_expand -unalias** {**all** | **none** | **dynamic**} You must choose one of the following: **all**, **none**, or **dynamic**. If you do not use **-unalias**, **-unalias dynamic** is the default behavior. **all** Resolves host aliases before printing the expansion. **none** Does not resolve any host aliases. **dynamic** Resolves dynamic host aliases that are introduced when AB_AGENT_COUNT is set to a value greater than **1** For information about host aliases and AB_AGENT_COUNT, see "AB_AGENT_COUNT sets number of agents on SMPs" on page 103.

A note on file skew

The concept of skew refers to an unbalanced load among the partitions of a multifile (or among the partitions of a dataflow). Indicators of skew are produced by the utilities **m_ls**, **m_du**, and **m_df**. Skew is defined as follows:

Assume that:

- For a particular flow or file, there are *k* partitions.
- There are *total* bytes in all the partitions.

Then,

average = *total* / *k*

Let *max* be the size of the partition with the most bytes.

Then, the skew for a partition with *n* bytes is:

(*n* - *average*) / *max*

The skew is scaled from **0** to **100** to give a percentage.

- As usual, if *n* is *average*, the skew is **0**.
- If *n* is less than *average*, the skew becomes less than **0**, down to a minimum of **-100%** (because *average* is never greater than *max*).
- If *n* is greater than *average*, the skew approaches **100%** (*n* is never greater than *max*, either).

The overall skew is the maximum skew of all the partitions.

By the way, the sum of the skews is **0%**.

NOTE: Skew may be computed on bytes (as shown above) or on some other quantity such as blocks.

For information on reporting skew in executing graphs, see "Skew and its measurement" on page 67.

Multifile management at run time

Ad hoc multifiles

You can build parallel **mp** file components (**ifile**, **ofile**, and **iofile**) "on the fly" by explicitly naming a set of serial files. The names may be pathnames or URLs. The syntax for declaring file components is:

mp ifile *name url* [*url* ...] [**-flags** *flags*] [**-m**[**ode**] *mode*]
mp ofile *name url* [*url* ...] [**-flags** *flags* [**-m**[**ode**] *mode*]
mp iofile *name url* [*url* ...] [**-flags** *flags*] [**-m**[**ode**] *mode*]

For example, consider this **mp** command line:

```
mp ifile my-input \
   file://host1/user1/a.dat \
   file://host2/user2/b.dat
```

This line declares **my-input** to be a two-partition file: the data in the first partition is the data in **//host1/user1/a.dat**, and the data in the second partition is the data in **//host2/user2/b.dat**.

Hostname aliasing

The situation may arise where a host that contributes a partition of a multifile system is not available when used in an application layout. This will normally render the layout or the multifile system unusable. However, it may be possible, through network-mounted disks or dual-ported disk controllers, that some other computer can act as a surrogate for the missing host.

The surrogate host would need to have access to the same disks, mounted under the same names, as the missing host, but this is common in fault-tolerant systems or systems with redundant data access.

In this situation, the Ab Initio system provides a mechanism for declaring a hostname alias for the missing host. An alias simply specifies that when the program accesses a certain host name, the name of the surrogate host should be substituted. You can define any number of such aliases, and they are in effect for the duration of an application only.

You define a set of aliases by creating a text file containing the following:

```
# This is a comment and is ignored
hostname alias
hostname alias
...
```

For example:

```
# This file allows hosts n20 and n21 to stand in for
# hosts n0 and n1, which exploded late last night.
n0 n20
n1 n21
```

Before running the application, you need to set the environment variable HOST_ALIAS_FILE to the pathname of the alias file:

```
export HOST_ALIAS_FILE=alias-file-name      [ksh]
setenv HOST_ALIAS_FILE alias-file-name      [csh]
```

At run time, whenever the application refers to **n0**, the host **n20** will be used instead, as will host **n21** for **n1**. The substitution includes access to multifile systems that have data partitions on either host **n0** or host **n1**.

3>

Metadata management

Ab Initio metadata describes data formats and computations. This chapter describes Co>Operating System utilities for managing, examining, and generating metadata.

See the following document:

- *Data Manipulation Language Reference*

for more information on metadata.

Metadata catalog facility

The Co>Operating System's catalog facility provides a systematic means of associating metadata with data. It serves primarily to support DML lookup tables. The catalog facility consists of a set of commands for manipulating catalogs from a shell.

Concepts

A catalog defines a set of tables, each of which has:

- A unique name
- A primary key
- A data URL
- A metadata URL

There is a central, default catalog, which may be read-only. You can designate other catalogs for development purposes or on a per-application basis.

Manipulating catalogs

The catalog facility provides shell commands that:

- Create a catalog
- Delete a catalog
- List a catalog
- Add a table to a catalog
- Remove a table from a catalog
- Find a catalog entry given the table name
- Find a catalog entry given the data URL

In addition, the shell commands provide the ability to export catalogs to text files and import catalogs from text files. This allows you to put catalogs under a line-oriented source control system (e.g., RCS or CVS) for versioning.

Designating a catalog

By default all commands apply to the central, default catalog. The environment variable AB_CATALOG can be set to the path of an alternate catalog, if desired. In addition, all commands take a **-catalog** argument which may specify an alternate catalog. If both are used, the command argument overrides the environment variable.

Commands

m_mkcatalog -catalog *catalog_url*

Create a catalog.

m_rmcatalog -catalog *catalog_url*

Remove a catalog.

m_lscatalog -catalog *catalog_url* **[[-table** *name* **| -data** *data_url* **][-table** *name* **| -data** *data_url* **]...]**

List the catalog in text form. If table names are specified, list the entries for those tables. If data URLs are specified, list the entries with those data URLs. If neither table names nor data URLs are specified, list all entries.

m_catalog_add -catalog *catalog_url* **-table** *name* **-key** *key* **-data** *data_url* **-metadata** *metadata_url*

Add an entry to the catalog. The table name, key, data URL, and metadata URL must all be supplied.

m_catalog_delete -catalog *catalog_url* **[[-table** *name* **| -data** *data_url* **][-table** *name* **| -data**
*data_url***]...]**

Remove entries from the catalog as specified by table names or data URLs.

m_catalog_export -catalog *catalog_url*

Produce a text representation of the specified catalog on standard output. This representation of the catalog is (intentionally) not specified, but is guaranteed to be suitable for use with line-oriented versioning tools, such as **diff**, RCS, and CVS.

m_catalog_import -catalog *catalog_url*

Populate the specified catalog using the text representation of a catalog provided on standard input. If the catalog exists, it must be empty.

Examples

Example 1

```
m_mkcatalog -catalog test1
```

Example 2

```
m_catalog_add -catalog test1 -table january_claims \
-data file://control/mfs/monthly/jan.dat \
-metadata file://control/mdr/claims/monthly.dml \
-key claim_id
```

Example 3

```
m_catalog_add -catalog test1 -table customers \
-data mfile://fr1n01/CUSTOMERS \
-metadata file://control/mdr/dw/customers.dml \
-key SSN
```

Example 4

```
m_catalog_export -catalog test1
```

Example 5

```
m_lscatalog -catalog test1

Table:     january_claims
 Key:     claim_id
 Data URL:   file://control/mfs/monthly/jan.dat
 Metadata URL: file://control/mdr/claims/monthly.dml
Table:     customers
 Key:     SSN
 Data URL:   mfile://fr1n01/CUSTOMERS
 Metadata URL: file://control/mdr/dw/customers.dml
```

Example 6

```
m_catalog_add -catalog test1 -table test_claims \
-data file://control/mfs/test/test.dat \
-metadata file://control/mdr/claims/monthly.dml \
-key claim_id
```

Example 7

```
m_lscatalog -catalog test1

Table:     january_claims
 Key:     claim_id
 Data URL:   file://control/mfs/monthly/jan.dat
 Metadata URL: file://control/mdr/claims/monthly.dml
Table:     customers
 Key:     SSN
 Data URL:   mfile://fr1n01/CUSTOMERS
 Metadata URL: file://control/mdr/dw/customers.dml
Table:     test_claims
 Key:     claim_id
 Data URL:   file://control/mfs/test/test.dat
 Metadata URL: file://control/mdr/claims/monthly.dml
```

Example 8

```
m_rmcatalog -catalog test1
```

Example 9

```
m_catalog_import -catalog test1 < test1.catalog-save
```

Example 10

```
m_lscatalog -catalog test1

Table:     january_claims
 Key:     claim_id
 Data URL:   file://control/mfs/monthly/jan.dat
 Metadata URL: file://control/mdr/claims/monthly.dml
Table:     customers
 Key:     SSN
 Data URL:   mfile://fr1n01/CUSTOMERS
 Metadata URL: file://control/mdr/dw/customers.dml
```

Example 11

```
m_rmcatalog -catalog test1
```

4>

Job management

An Ab Initio application executes within a framework called a job. A job is created by the **mp job** command, which is usually the first line of the script that defines an application. The job includes all the processes, disk I/O, and data transfers needed to complete the task, as well as the monitoring, checkpointing, and other task management activities specified in the application.

The application script executes on some host and uses some directory as its working directory. If its layout specifies multiple hosts/directories, the job includes multiple copies of the program running in these different locations. The Co>Operating System manages all the processes and data flows. Informational messages from the Co>Operating System identify the job with the name given to it by the mp job command.

This chapter presents the concepts and commands you need to configure the execution environment and to manage Ab Initio jobs. Job management includes:

- Execution
- Checkpointing, termination, and recovery
- Monitoring

Program layouts and remote connections

Every component of an application, whether a dataset or a program, has a layout. The layout defines the location(s)–*hostname*/*pathname*–where that component occurs. Some applications use one layout throughout, for both the data components and the program components. Others use several layouts and repartition data as needed for processing by more or fewer hosts than previous components.

A layout is a list of URLs, each specifying a host and a working directory on that host for one of the partitions of a dataset or of a program component. For example, the layout used to run a program on four hosts (**host1** through **host4**), using **/tmp** as the working directory, is defined to include these four URLs:

```
//host1/tmp
//host2/tmp
//host3/tmp
//host4/tmp
```

To execute an Ab Initio application, the URLs of its layout(s) must describe locations that are valid for this use. That is,

- The hosts must have the Ab Initio software installed.
- The working directories must already exist.
- The working directories must be readable and writable from the user's login account.

The Co>Operating System may be configured to use any of four methods for remote connections: **rsh**, **rexec**, **telnet**, and /or **rlogin**. The default method is **rsh**.

Beforehand, the environment must be configured for **rsh** by listing the hosts that will be used for Ab Initio applications. The system administrator can list them centrally in **/etc/hosts.equiv**, or users can list them in individual .**rhost** files.

If **rsh** is not available or if **rsh** requires a password, then the recommended alternate is **rexec**. If neither **rsh** nor **rexec** is available, then **telnet** is preferred. The **rlogin** method is the last resort, and should not be used on some systems (e.g., HP-UX) for Ab Initio connections.

Configuring the environment

A set of environment variables controls the operation of Ab Initio software. Most of these are set by the system administrator, although individual users may have occasion to set or reset them locally.

The command **m_env** displays the current settings of the Ab Initio environment variables. Invoke **m_env** with the option **-h** for added help.

```
    m_env -h
```

Recall (from page 2) that at run time, environment variables are passed downward only. When a script is invoked, its environment variable settings will initially be those of the shell that invoked it. If the script changes any of the values, those values are changed only for the course of the application. They are not changed in the invoking shell.

This section lists only the environment variables that are most often set by users. For a complete list of environment variables, see "Notes on administration" on page 133.

Ab Initio software location

As noted in Chapter 1> "Overview"on page 1, two environment variables must be set before any Ab Initio commands can be used. These settings enable the users local environment to locate the installed Ab Initio software.

AB_HOME=path	The home directory of the installed Ab Initio software, usually **/usr/local/abinitio**. It is not necessary to set this variable on the remote hosts used by the application. The Co>Operating System will set it on all remote hosts to the same value it has on the host where the application script runs.
PATH=path	Search the **bin** directory under the home of the installed Ab Initio software; the value is usually **$AB_HOME/bin:$PATH**.

System behavior

Set these environment variables if you want a value different from the default. These are usually set centrally by the system administrator, but a user may wish to reset them when running a particular application.

AB_TIMEOUT=*seconds*	
	The time-out interval for certain operations, such as starting up a remote process. Default is **30** seconds.
AB_MAX_RECORD_BUFFER=*bytes*	
	Maximum buffer size that certain parts of the system will use to hold a record. Default is **5000000** (5 million) bytes. Note: Incorrect metadata or corrupt data may cause data to appear to the system as an unreasonably large record. When the space needed to buffer a record exceeds the value of AB_MAX_RECORD_BUFFER, the system reports an error and aborts execution.
AB_NICE=*priority*	
	Run jobs on remote hosts at the specified priority. The values reflect UNIX priorities, which range from **-20** (highest) to **19** (lowest). A user's shell typically runs at priority **0**, and only the superuser can increase priorities (i.e., move to lower numerical values). As a result, the user can set AB_NICE to values from **1** (just a little nice) through **19** (exceedingly deferential). If an unattainable value is specified, the processes will run at the default priority. Note to Korn shell users: The shell transparently reduces the priority of any job that runs in the background (and the priority of any processes directly or indirectly created by such a job). This can degrade performance. To avoid this behavior, set the **ksh** option **bgnice** to +**o**.

Special Ab Initio facilities

These facilities are described in more detail elsewhere in this manual.

AB_HOST_ALIAS_FILE=*path*	
	File containing hostname aliases. See page 29.
AB_CATALOG=*path*	
	Location of user-created metadata catalogs. See page 32.
AB_REPORT=*keyword*	
	Monitor the current job and produce reports. See "AB_REPORT settings" on page 58 for keyword values.

Querying the environment with m_env

The **m_env** utility, when invoked with no options, reports the current settings of some of the standard configuration variables. It has several options to report specific configuration information.

-help	
	A brief summary of the arguments to **m_env**.
-v or **-version**	
	Reports the Ab Initio software version.
-doc	
	Lists the Ab Initio documentation that describes the current software version.
-h or **-describe**	
	Descriptions of some of the standard configuration variables.
-d or **-debug-info**	
	Descriptions of some of the debugging configuration variables.
-n *host-or-alias* or **-node-info** *host-or-alias*	
	Settings that will be used when Ab Initio is executing components on the named host (or aliased group of hosts).
-g *config_var* [*host-or-alias*] or **-get** *config_var* [*host-or-alias*] or **-get-value** *config_var* [*host-or-alias*]	
	Reports the value of the configuration variable *config_var*. If *host-or-alias* is supplied, then it is the value that will be used when Ab Initio is executing components on the named host (or aliased group of hosts). This option is generally only useful if *config_var* is initialized either in the environment or in one of the configuration files.

Execution

An Ab Initio application is a shell script. It is invoked from a shell like any other script:

```
    my_application
```

To abort the application at any time:

Press C while holding down the Control key.

Pre-flight checklist

Make sure that environment variables are set appropriately. See page 134 for a complete list. The command **m_env** is useful for checking the current settings.

- AB_HOME and PATH must be set in the user's environment.
- If running on an IBM SP2, AB_HOST_INTERFACE should be set to cause applications to use the high-speed switch rather than the Ethernet for interprocessor data movement. The system administrator may have done this centrally.
- Customize the settings of other environment variables as desired. (Many of these may have been set centrally by the system administrator, and several have default values.)

Make sure that the necessary resources exist and are available to the application:

- All processing hosts used in any of the application's layouts, including program component layouts and multifile layouts, must be up and running. (But note the facility for hostname aliasing, page 29, as a workaround for unavailable hosts.)
- Input data files must exist (of course), though output files need not. If an output file is to be a multifile, the multidirectory in which it will be created must exist before execution.
- Metadata files referenced in **mp** metadata commands or used as arguments to program components must exist. Metadata files, which usually have the filename extension **.dml**, describe data formats and data computations and transformations.

Enabling other services

Any desired Co>Operating System services, such as checkpointing, monitoring, and debugging, are enabled before invoking an application. The facilities are often invoked by **mp** commands in the application script. You may also use environment variables to enable these services or to add to the script’s specification.

The *Ab Initio Shell Development Environment User’s Guide* describes the **mp** commands that enable Co>Operating System services from within the script. The shell-level **m_***name* commands and environment variables that control monitoring, checkpoint/recovery, and debugging from the shell are described below in this chapter.

What happens at run time

An Ab Initio application is a set of **mp** commands, beginning with **mp job** and ending (usually) with **mp run**. In between are commands that identify the program components and indicate the flow of data from one to the next. Thus, the **mp** script usually defines `and` runs a job.

Creating the program graph

When a script is invoked, the **mp job** command executes. At this point, the system creates two files in the current working directory:

- *jobname*.**job**.

  As the rest of the script is read, a text representation of the application being defined is placed here. This file is a text file, but it is strongly recommended you do not edit it.

- **.abinitio-current-job**.

  This file contains jobname; it enables the system to know the name of the current job.

Notice that if two or more **mp** jobs are running in the same directory at the same time, one job will overwrite the other's **.abinitio-current-job** file. To avoid this problem, use the environment variable AB_JOB. When AB_JOB is set, all **mp** commands use its value as the name of the current **mp** job, ignoring the name stored in **.abinitio-current-job**. By setting AB_JOB, you enable two or more mp jobs to run in the same directory at the same time, as long as their job names differ from one another. This variable is described in more detail in the *Ab Initio Shell Development Environment User's Guide*.

Executing the graph

When the script reaches the **mp run** command, the Co>Operating System reads the defined application and begins execution in a process on the host computer.

The host process forks to spawn the Host State Database, which is actually another process. The host process also uses remote process invocation (**rsh**, by default) to start up agent processes on all hosts that the application uses. These agents, in turn, spawn Agent State Database processes on their respective hosts, as well as spawning the processes that actually execute the components of the application.

Log files

The Ab Initio system uses a sophisticated transaction-based system to implement all file and flow operations. The State Databases generate log files on the host system and on each computer used by the application. These files record the changes the job makes to files, flows, and processes. If a job terminates abnormally, the system uses these files to undo the changes it has made, and thus cleanly restore the system to an earlier state. See page 49. If the job exits successfully, the log files are deleted.

Environment variables

The original host process invoked by **mp run** (often called the driver process) also detects the values of environment variables for the job. Initially, the values are those that the system administrator has set centrally as part of the system configuration. See page 134. These configuration settings may be overridden by the settings of environment variables in a particular shell.

Multiphase execution and checkpointing

An Ab Initio application may be designed to execute in sequential phases, with or without checkpointing, which means saving the state of the graph to disk between phases.

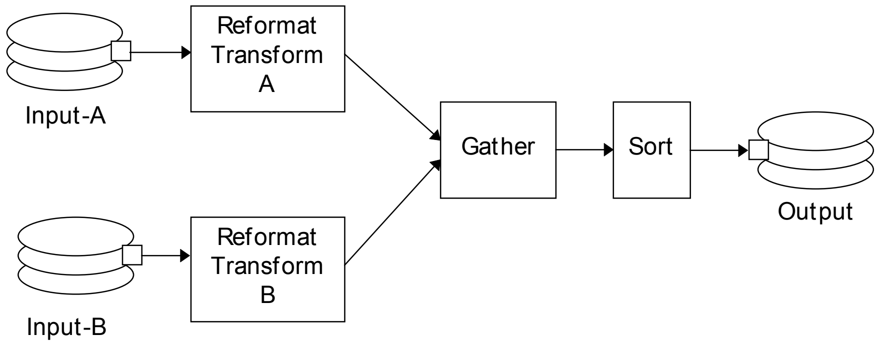
Phased execution is enabled from within the application, if the script developer has inserted the command **mp phase** or **mp checkpoint** between one component and another.

This section describes phased execution briefly, even though it is not controlled by the user who is executing the program. Phasing makes a difference in how the application uses system resources, often trading off performance for safety. Also, phasing - and particularly checkpointing - affects your actions in the event of abnormal termination of the program.

Multiphase execution

An application may consist of discernible phases of execution. This is usually done to guarantee that sufficient resources will be available for an especially demanding part of the application.

For instance, consider this application:



For resource reasons, it might be advisable to complete the two transforms before beginning the sort, since a sort tends to be memory-intensive. If so, the script developer will have inserted the command **mp phase** just before the **mp sort** command. Phasing inhibits pipeline parallelism but guarantees that resource-intensive stages will not compete with each other.

Checkpointing

In the example just shown, you might want to run three processing phases sequentially as protection in the event of failure. Should the system or the application fail at any point, it would be helpful if the results of the previous steps were saved–or checkpointed–so that those steps need not be repeated.

For this purpose, the script developer may have inserted **mp checkpoint** commands between the two transform components and before the gather. If so, the application runs as follows:

- First, the Reformat-Transform-A program runs to completion, reading and transforming the contents of the file Input-A.
- The component Reformat-Transform-A writes its output to the flow that connects it to Gather, but the Gather program does not read it until a later phase. Instead, the Ab Initio system automatically inserts a temporary file at the phase boundary to buffer data that is bound for programs that run in later phases.
- When the first phase completes, the second begins: the Reformat-Transform-B program runs to completion, reading and transforming the contents of the file Input-B. Again, the output is written to disk rather than being passed to the Gather component.
- In the third and final phase, the data from the two temporary files is gathered together, sorted, and written to the output file.

Like phasing, checkpointing trades some speed for safety. Not only do the phases execute separately (which inhibits pipelining), but extra I/O is performed between phases. However, in the event of failure, the application can restart from the most recent checkpoint instead of from the beginning.

Abnormal termination and recovery

An Ab Initio job is considered completed when all the processes associated with that job complete. These include the process on the host system that executes the script and all processes the job has started on remote notes.

If any of these processes terminates abnormally, the Co>Operating System terminates the entire job and then cleans up the mess as much as possible.

- In the case of software error or user Control-C command, the Co>Operating System can perform an automatic rollback, thus restoring all files, flows, and processes to their initial state or to their state at the most recent checkpoint.
- In the case of the crash of a host or the native operating system, the user may need to perform a manual rollback of the failed program.

Because the Co>Operating System keeps track of all file changes and other changes a job has made, you need not investigate how far the job had gone or manually modify an application to pick up where it left off.

Recovery files

When a job does not complete normally, it leaves a file in the working directory on the host system with the name *jobname***.rec**. This file contains a set of pointers to the log files on the host and on every computer. The log files enable the Co>Operating System to roll back the system to its initial state (or to its state as of the most recent checkpoint).

The log files are placed in log subdirectories that are created when the application starts and deleted when the application successfully completes. The default locations of these subdirectories are:

- On the host system: **/var/abinitio/host/***unique-id*, where *unique-id* is a sequence of characters unique to each execution run of the application.
- On each remote host: **/var/abinitio/vnode/***unique-id*, where *unique-id* is the same as that used on the host system.

NOTE: The system administrator may place the log directories in other locations when the Ab Initio system is installed.

Automatic rollback

If the application encounters a software failure (for example, one of the processes signals an error or the operator aborts the application), all hosts and their respective files will be rolled back to their initial state, as if the application were not run at all. The files will be as they were at the start, all temporary files and storage will be deleted, and all processes will be terminated. If the program contains checkpoint commands, the state restored is that of the most recent checkpoint.

Specifically, the Ab Initio system will:

- Kill all processes running on all hosts, including control processes and processes that constitute the partitions of a parallel program.
- Cleanly shut down all data flows.
- Roll back the effects of all file changes.
- Report the state of the system.
- Exit.

When a job has been rolled back, you recover it simply by rerunning it. Of course, the cause of the original failure may also repeat itself when the failed job is rerun. You will have to determine the cause of the failure by investigation or by debugging.

When a checkpointed application is rerun, the Ab Initio system performs a "fast-forward" recapitulation of the successful phases. During this recapitulation, no programs run and no data flows, i.e., the phases are not actually repeated (although the monitoring system cannot detect the difference between the recapitulation and an actual execution). When the recapitulated phases are completed, the failed phase is re-run.

Manual rollback

However, it may not always be possible for the Co>Operating System to restore the system to an earlier state. For example, a failure could occur because a host or its native operating system crashed. In this case, it is not possible to cleanly shut down flow or file operations, nor to roll back file operations performed in the current phase. In fact, it is likely that stray files (intermediate temporaries) will be left lying around.

To complete the cleanup and get the job running again, you must perform a manual rollback. For this, you use the command **m_rollback**.

m_rollback [**-d**] [**-i**] [**-h**] *recoveryfile*

-d	Delete the job along with its recovery file and any log files it created.
-i	Display the state of the job and prompt the user whether the job should be deleted.

If the **-i** option is not used, jobs that have reached their first checkpoint will be rolled back to the checkpoint. Jobs that do not include checkpoints or that did not reach their first checkpoint will be deleted.

The default behavior of **m_rollback** when called on a running job is to display a warning message then exit.

The shell argument **-kill** causes **m_rollback** to attempt to kill the job. The command waits for up to $AB_TIMEOUT seconds to verify that the job has been killed. The default value for $AB_TIMEOUT is **30** seconds.

Examples

Example 1

Action of **m_rollback** for a job that has no checkpoints or that has not reached a checkpoint.

```
m_rollback my-job.rec

ABINITIO: Recovery of job from recovery file
"/l8b/user_name/my-job.rec" in progress

ABINITIO: Job closed and recovery file deleted
```

Example 2

Action of **m_rollback** for a job that has reached a checkpoint.

```
m_rollback my-job.rec

ABINITIO: Recovery of job from recovery file
"/l8b/user_name/my-job.rec" in progress

ABINITIO: Job rolled back to phase 1

ABINITIO: Job may be continued from last
checkpoint by:   re-running.

ABINITIO: Job may be closed and recovery file
deleted by: m_rollback -d
/l8b/user_name/my-job.rec.
```

Example 3

```
m_rollback -d my-job.rec

ABINITIO: Recovery of job from recovery file
"/l8b/ user_name /my-job.rec" in progress

ABINITIO: Job closed and recovery file deleted
```

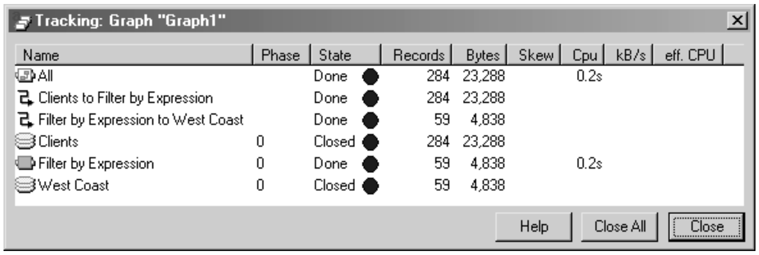
Tracking graph execution in the GDE

To display tracking information for a job run from the GDE, right-click in the workspace of the graph that you want to track, and then click on **Tracking Detail**. The Tracking dialog box appears containing the results of the last graph execution. Execute the graph again to display more recent results.

You can track all or any combination of individual flows and components within a graph. To do so, select the flows and components you want to track, right-click, and then click on **Tracking Details**. A Tracking dialog box appears for each selection.

You can save the tracking information of the last execution of the graph when you save it. To do so, select **Run** > **Settings**, which opens the Run Settings dialog box. Click the Tracking tab of the Run Settings dialog box. Select **Save Tracking Data with Graph** and click **OK**.

The following is the Tracking dialog box of the graph in the online tutorial. (See **Help** > **Tutorial**.)


Tracking: Graph "Graph1"

Name	Phase	State	Records	Bytes	Skew	Cpu	kB/s	eff. CPU
All		Done ●	284	23,288		0.2s		
Clients to Filter by Expression		Done ●	284	23,288				
Filter by Expression to West Coast		Done ●	59	4,838				
Clients	0	Closed ●	284	23,288				
Filter by Expression	0	Done ●	59	4,838		0.2s		
West Coast	0	Closed ●	59	4,838				

The Tracking dialog box contains the following execution information for the graph:

Name	The name of the element for which the row contains execution information. An element can be a dataset component, program component, table component, graph, subgraph, or flow.
Phase	The phase in which the element executed. For information about phases, see **mp checkpoint** on page 174.
State	For a dataset component, unopened, open, closed, or error. For other elements, unstarted, run, done, or failed.
Records	Number of records associated with the element.
Bytes	Number of bytes associated with the element.
Skew	See "Skew and its measurement" on page 67.

Cpu	Number of CPU cycles per unit for components.
kB/s	Number of kilobytes of all input flows.
eff. CPU	Number of CPU seconds per second. On a multi-processor CPU, this number can be greater than one. The maximum value is equal to the number of processors on the machine. The "eff." stands for “Effective.” The number is a measure of the efficiency of the graph..

Tracking the execution of deployed scripts

The amount of reporting done by a graph deployed from the GDE as a script is controlled by specifying settings to the AB_REPORT configuration variable either in the environment in which the script is run, or in the script itself.

Simple example

The following simple exercise, using the **Intro.mp** example graph, will demonstrate some of the basic features of the reporting mechanism.

1. Choose **Help** > **Examples** in the GDE menu.
2. Double-click on the DML directory name displayed in the window.
3. Double-click on the name of the graph **Intro.mp**.
4. Choose **Run** > **Deploy** > **As Script**.

   This causes the graph to be written as a script in the Host Directory specified in your Run Settings (or to the $RUN directory, if you are working in a sandbox). When the script is written, a **Deploy Job -- Done** dialog appears. Click Details for the location of the deployed script.
5. On the command line in the script's Korn shell environment, type the following:

```
export AB_REPORT='monitor=60 processes scroll=true file=report_file.txt'
```

   Note that if you are using a version of the GDE prior to version 1.12, you cannot set AB_REPORT in the shell as shown, because the AB_REPORT setting in the deployed script cannot be overridden. Load the deployed script into the text editor of your choice, and edit the script's AB_REPORT line so that it reads as follows:

```
AB_REPORT='monitor=60 processes scroll=true file=report_file.txt'
```

6. Run the script.

When the script has completed execution, the file **report_file.txt** will have the following content:

```
----------------------------------------------------------------------------------
Jun 23 10:34:11   Phase 0 started (1 second)                                     ①
                        CPU Time  Status Skew Vertex
                           0.060 [ 1:  ]   0% Dedup_Sorted
                           0.050 [ 1:  ]   0% Filter_by_Expression
                           0.070 [ 1:  ]   0% Reformat
                           0.110 [ 1:  ]   0% Rollup
----------------------------------------------------------------------------------
       Data Bytes          Records      Status      Skew Flow Vertex   Port      ②
            1,148               14 [    :   1:    ]   0% Flow_1 Dedup_Sorted in
              820               10 [    :    :   1]   0% Flow_2 Dedup_Sorted out
                0                0 [    :   1:    ]   0% Flow_5 Rollup       in
                0                0 [    :   1:    ]   0% Flow_6 Rollup       out
----------------------------------------------------------------------------------

----------------------------------------------------------------------------------
Jun 23 10:34:11   Phase 0 ended (1 second)                                       ③
                        CPU Time  Status Skew Vertex
                           0.080 [  : 1]   0% Dedup_Sorted
                           0.090 [  : 1]   0% Filter_by_Expression
                           0.110 [  : 1]   0% Reformat
                           0.140 [  : 1]   0% Rollup
----------------------------------------------------------------------------------
       Data Bytes          Records      Status      Skew Flow   Vertex        Port
            1,148               14 [    :    :   1]   0% Flow_1 Dedup_Sorted in    ④
              820               10 [    :    :   1]   0% Flow_2 Dedup_Sorted out
              820               10 [    :    :   1]   0% Flow_2 Filter_by_Expression in
              656                8 [    :    :   1]   0% Flow_3 Filter_by_Expression out
              656                8 [    :    :   1]   0% Flow_3 Reformat           in
              736                8 [    :    :   1]   0% Flow_4 Reformat           out0
            1,148               14 [    :    :   1]   0% Flow_5 Rollup             in
               96                6 [    :    :   1]   0% Flow_6 Rollup             out
----------------------------------------------------------------------------------
```

There are four sections in this report:

1. Describes component processes at the beginning of phase 0 of the graph. This part of the report is produced by the **processes** setting in AB_REPORT.
2. Describes the graph flow states at the beginning of graph execution. This part of the report is produced by the **monitors=60** setting, which specifies that information about flow states be shown every 60 seconds.
3. Describes component processes at the end of phase 0 (the only phase) of the graph. This part of the report is also produced by the **processes** setting in AB_REPORT.
4. Describes the graph flow states at the end of graph execution. Since the duration of graph execution was less than 60 seconds, graph process information is displayed only at the beginning and end of execution. This part of the report is also produced by the **monitors=60** setting.

Processes section of report

The following information is shown in the **processes** sections (**1** and **3**):

CPU Time	Number of CPU seconds used by the component. See “processes” on page 60.
Status	Number of **[***running***:***finished***]** partitions of the component.
Skew	See “Skew and its measurement” on page 67.
Vertex	Name of the component.

Flows section of report

The following information is shown in the flows sections (**2** and **4**):

Data Bytes	Bytes in all the data partitions of the flow.
Records	Records in all the data partitions of the flow.
Status	Number of **[***unopened***:***open***:***closed***]** partitions in the flow. Flows start in the unopened state, then progress to the open state, and finally to the closed state.
Skew	See “Skew and its measurement” on page 67.
Flow	Name of the flow.
Vertex	Name of the component to which the flow is attached.
Port	Name of the port to which the flow is attached.

Reporting on flows only

The same default AB_REPORT setting in **Intro.ksh**, but without **processes** specified, would produce a report like the following:

```
--------------------------------------------------------------------------------
Jun 23 11:39:42   Phase 0 started (1 second)
       Data Bytes         Records      Status      Skew Flow Vertex   Port
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
Jun 23 11:39:43   Phase 0 ended (2 seconds)
       Data Bytes         Records      Status      Skew Flow Vertex   Port
            1,148              14 [    :    :   1]   0% Flow_1 Dedup_Sorted in
              820              10 [    :    :   1]   0% Flow_2 Dedup_Sorted out
              820              10 [    :    :   1]   0% Flow_2 Filter_by_Expression in
              656               8 [    :    :   1]   0% Flow_3 Filter_by_Expression out
              656               8 [    :    :   1]   0% Flow_3 Reformat             in
              736               8 [    :    :   1]   0% Flow_4 Reformat             out0
            1,148              14 [    :    :   1]   0% Flow_5 Rollup               in
               96               6 [    :    :   1]   0% Flow_6 Rollup               out
--------------------------------------------------------------------------------
```

Only flow information is shown in this report.

Changing the AB_REPORT setting

To set the default for AB_REPORT, GDE versions of 1.12 and later write the following lines in a deployed script:

```
AB_REPORT=${AB_REPORT:-'monitor=60 processes scroll=true'}
```

—which allows the script’s AB_REPORT setting to be overridden in the environment.

Changing AB_REPORT in GDE versions prior to 1.12

Versions of the GDE prior to 1.12 write the AB_REPORT default setting as

```
AB_REPORT='monitor=60 processes scroll=true'
```

in deployed scripts. If you wish to change the AB_REPORT settings for such a deployed script, you must do one of the following:

- Re-edit the script line in which AB_REPORT is set.

  There are two lines which occur near the beginning of the script. They look like this:

  ```
  export AB_REPORT
  AB_REPORT='monitor=60 processes scroll=true'
  ```

  You should edit the second line to add or change the AB_REPORT settings you want.

- Completely comment out the script’s AB_REPORT lines, and set AB_REPORT as desired in the shell environment in which you invoke the script.

  With this method, you can set and export AB_REPORT directly on the command line, or in a script that invokes the deployed script.

AB_REPORT settings

The two basic aspects of graph performance you can monitor are:

- flows

  Specified by **flows** (and **interval** to specify reporting intervals)

  Alternatively, you can specify **monitor**, which is effectively a combination of the **flows** and **interval** settings.

- processes

  Specified by **times** or **processes**.

The full list of AB_REPORT settings is given in the following sections.

file=*pathname*

Allows you to specify a file destination for the contents of the report, rather than have it displayed to the screen when the script is run.

With **file** specified, subsequent runs of the script cause the report information to be *appended* to the already-existing file; the report file is *not* overwritten.

file-percentages

For flow monitoring, specifies a percentage-done ("%") column showing the progress of file-input flows.

Specifying **file-percentages** in AB_REPORT causes an additional column, labeled "(%)", to be displayed in the report. At each given report interval, the figure in this column shows how much of an Input File (or Lookup File) has been directly consumed by the component reading from it.

Note that file components are never directly represented in report listings. The figures for input file consumption are thus reported against the input flow of the component consuming the file. For example, the following report fragment:

```
---------------------------------------------------------------------------------
Jun 26 11:39:57   Phase 0 running (10 seconds)
                        CPU Time  Status Skew Vertex
                           0.160 [ 2:  ]   0% Dedup_Sorted
                           5.488 [ 2:  ]   0% Sort
---------------------------------------------------------------------------------
       Data Bytes      (%)          Records      Status      Skew Flow Vertex   Port
       22,269,311     (20%)         293,010 [    :   2:    ]   0% Flow_1 Sort     in
                0                         0 [    :   2:    ]   0% Flow_2 Sort     out
                0                         0 [    :   2:    ]   0% Flow_2 Dedup_Sorted in
                0                         0 [    :   2:    ]   0% Flow_3 Dedup_Sorted out
---------------------------------------------------------------------------------


---------------------------------------------------------------------------------
Jun 26 11:40:06   Phase 0 running (20 seconds)
                        CPU Time  Status Skew Vertex
                           0.160 [ 2:  ]   0% Dedup_Sorted
                          10.745 [ 2:  ]   0% Sort
---------------------------------------------------------------------------------
       Data Bytes      (%)          Records      Status      Skew Flow   Vertex       Port
       33,403,901     (29%)         439,451 [    :   2:    ]   0% Flow_1 Sort         in
                0                         0 [    :   2:    ]   0% Flow_2 Sort         out
                0                         0 [    :   2:    ]   0% Flow_2 Dedup_Sorted in
                0                         0 [    :   2:    ]   0% Flow_3 Dedup_Sorted out
---------------------------------------------------------------------------------
```

—shows two successive 10-second intervals during the execution of a graph in which a Sort component is reading from an Input File. The per cent consumption of the Input File is reported (in the "(%)" column) against the consuming Sort component's **in** port.

flows

Specifies that flow information be displayed. Note that **monitor** has the same effect, with the addition that **monitor** can optionally be specified with a reporting interval.

interval=*n*

Specifies the interval (in seconds) at which report information should be displayed. Can be used with **flows**, **times**, or **processes**.

monitor[**=***n*]	This is a combination of **flows** and **interval**. *n*, if specified, is the desired report interval, expressed in seconds.
processes	Identical to specifying **times**. During execution, what is identified as a single graph "process" in a report may in fact represent multiple child processes accomplishing the work of a single entity. Under these circumstances, the reporting mechanism has to rely on a variety of platform-dependent and somewhat inconsistent means to obtain CPU time information, while the processes are executing. After process completion, the reporting mechanism is able to access accurate final CPU time data. For this reason, often the final reported time of a process will be found to have increased in comparison with the times reported during execution.
scroll={**true**|**false**}	Specifies that special formatting be applied to ensure that report listing lines are correctly formatted and scrolled in the console window. See "Display modes" on page 76. If **file** is specified, the default value for **scroll** is **true**; otherwise (if the report is being written to the terminal) the default value for **scroll** is **false**.
skew[**=***min_skew*[**:***plies*]]	Specifies that flow plies with skew whose absolute value exceeds the value *min_skew* be shown. The optional **:***plies*, which is 4 by default, specifies the maximum number of flow plies (partitions) to show. Specifying **all** or ***** for value causes all plies whose skew exceeds *n* to be shown. See "Skew and its measurement" on page 67.
spillage	This measurement is relevant only when flow buffering is in effect in a graph: it shows how much of the buffer contents is "spilled" (temporarily written) to disk in situations where the buffered data becomes too large to be held in memory. For process monitoring, specifying **spillage** causes additional "Maxcore" and "Spilled Bytes/Records" columns to be included in the report, giving information on maximum core usage, and the amount of data spilled to disk. See examples on page 64, page 65, and page 66.

For flow monitoring, specifying **spillage** causes an additional **Buffer** column to be included, showing bytes spilled for flow buffering. The letter "K", "M", or "G" is appended to this byte number if it is greater than 1000, indicating kilobytes, megabytes, or gigabytes, respectively. See examples on page 64, page 65, and page 66.

split-cpu

When specified with process monitoring, **split-cpu** produces separate columns in the report, showing the user and system CPU usage (i.e., time expended in user call and system call activity, respectively).

NOTE: The **split-cpu** option is not currently supported on all platforms. If separate system and user values are not available on your platform, the sum of the values is shown under user CPU, and system CPU is shown as 0.

summary

If **summary**=*filename* is specified as an AB_REPORT setting, the Co>Operating System creates a summary file at the end of the job, containing information about the job's elapsed time, runtime, and data volume.

Raw contents of a summary file

The contents of a summary file in their raw form look, for example, like this:

```
job-start 0 1998-11-24 15:03:22
component sel 0 finished 0.430
component ref 0 finished 0.430
flow sel.in 0 closed 5 43
flow sel.out 0 closed 3 26
flow ref.in 0 closed 3 26
flow ref.0 0 closed 3 26
phase-end 1 1998-11-24 15:03:25
component gen 0 finished 0.370
component hp 0 finished 0.380
component sink 0 finished 0.370
component sink 1 finished 0.390
flow gen.out 0 closed 20 517
flow hp.in 0 closed 20 517
flow hp.out 0 closed 12 276
flow hp.out 1 closed 8 241
flow sink.in 0 closed 12 276
flow sink.in 1 closed 8 241
phase-end 1 1998-11-24 15:03:26
```

(Note that two flow counts are reported for most flows: one for each port at either end.)

You usually will want to aggregate the raw contents of a summary file to get a useful summary. You can do this by processing the file as a dataset, using the record format file

$AB_HOME/include/summary.dml

which is described in "Summary reports" on page 73.

table-flows

Turns on monitoring for the flows used to distribute lookup tables.

times

Specifies CPU process time usage. See "processes" on page 60 and examples on page 62, page 64, page 64, and page 66.

Note that **times** and **processes** have an identical effect.

totals

Causes a "Total" line to be printed at the end of each process monitoring section (specified with the **processes** setting in AB_REPORT), giving total CPU usage for all processes.

verbose

Turns on verbose error reporting.

AB_REPORT settings examples

Following are some examples showing different combinations of AB_REPORT settings and the kind of report produced as a result.

Keywords "flows" and "times"

Specifying **flows** and **times** turns on flow and process CPU usage monitoring, with all other settings defaulted:

```
                  % export AB_REPORT="flows times"
                  % mp run
---------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
                        CPU Time  Status Skew Vertex
                          15.980 [ 2:  ]  19% Sort
                          22.900 [ 2:  ]   8% Sum
---------------------------------------------------------------------------
       Data Bytes          Records      Status      Skew Flow Vertex    Port
       22,110,000          221,100 [    :   2:    ]   0% f1   Sort      in
       44,220,000          442,200 [    :   2:    ]   0% f2   Sum       in
---------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keyword "flows"

Specifying **flows** alone turns on flow monitoring:

```
                  % export AB_REPORT="flows"
                  % mp run
---------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
       Data Bytes          Records      Status      Skew Flow Vertex    Port
       22,110,000          221,100 [    :   2:    ]   0% f1   Sort      in
       44,220,000          442,200 [    :   2:    ]   0% f2   Sum       in
---------------------------------------------------------------------------
```

Note again that (as in the above example) with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keyword "times"

Specifying times alone turns on process CPU usage monitoring:

```
                 % export AB_REPORT="times"
                 % mp run
---------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
                        CPU Time  Status Skew Vertex
                          15.980 [ 2:  ]  19% Sort
                          22.900 [ 2:  ]   8% Sum
---------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "times" "split-cpu"

Specifying **times** and **split-cpu** turns on process CPU usage monitoring, and causes two CPU columns to appear in the report, one showing user CPU usage, the other showing system CPU usage:

```
                 % export AB_REPORT="times split-cpu"
                 % mp run
--------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
                        Usr CPU  Sys CPU  Status Skew Vertex
                         13.580    2.400 [ 2:  ]  19% Sort
                         18.400    4.500 [ 2:  ]   8% Sum
--------------------------------------------------------------------------------
```

Keywords "flows" "times" "totals"

Specifying **flows**, **times**, and **totals** turns on flow and process CPU usage monitoring, and causes an additional line labeled "Total" to be displayed, showing total CPU usage and spillage for all processes.

```
                 % export AB_REPORT="flows times totals"
                 % mp run
---------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
                        CPU Time  Status Skew Vertex
                          15.980 [ 2:  ]  19% Sort
                          22.900 [ 2:  ]   8% Sum
                          38.880 [ 4:  ]      Total
---------------------------------------------------------------------------
       Data Bytes          Records      Status      Skew Flow Vertex   Port
       22,110,000          221,100 [    :   2:    ]   0% f1   Sort     in
       44,220,000          442,200 [    :   2:    ]   0% f2   Sum      in
---------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "flows" "spillage"

Specifying **flows** and **spillage** turns on flow monitoring, and causes an additional column to be displayed reporting bytes spilled for flow buffering. For flow buffers, a K, M, or G is appended to indicate kilobytes, megabytes, or gigabytes.

```
% export AB_REPORT="flows spillage"
% mp run
```

```
------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
       Data Bytes   Buffer          Records      Status      Skew Flow Vertex    Port
       22,110,000                   221,100 [    :   2:     ]  0% f1   Sort      in
       44,220,000      11M          442,200 [    :   2:     ]  0% f2   Sum       in
------------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "times" "spillage"

Specifying **times** and **spillage** turns on process CPU usage monitoring, and causes additional **Maxcore** and **Spilled Bytes/Records** columns to report max-core usage and the amount of data spilled to disk.

```
% export AB_REPORT="times spillage"
% mp run
```

```
----------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
        Maxcore  Spilled Bytes Records  CPU Time  Status Skew Vertex
                                          15.980 [ 2:  ]  19% Sort
            66%      3,000,000  30,000    22.900 [ 2:  ]   8% Sum
----------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "flows" "times" "spillage"

Specifying **flows**, **times**, and **spillage** produces a combination of the two previously-illustrated reports: both flow monitoring and process CPU usage monitoring are turned on, and extra columns are added to the report to show bytes spilled for flow buffering and to report max-core usage and the amount of data spilled to disk.

```
% export AB_REPORT="flows times spillage"
% mp run
```

```
------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
        Maxcore  Spilled Bytes Records  CPU Time  Status Skew Vertex
                                          15.980 [ 2:  ]  19% Sort
            66%      3,000,000  30,000    22.900 [ 2:  ]   8% Sum
------------------------------------------------------------------------------
       Data Bytes   Buffer          Records      Status      Skew Flow Vertex    Port
       22,110,000                   221,100 [    :   2:     ]  0% f1   Sort      in
       44,220,000      11M          442,200 [    :   2:     ]  0% f2   Sum       in
------------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "flows" "times" "file-percentages"

Specifying **flows**, **times**, and **file-percentages** turns on flow and process CPU usage monitoring, and adds a percentage-done (**%**) column showing the progress of file-input flows.

```
    % export AB_REPORT="flows times file-percentages"
    % mp run
------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
                        CPU Time  Status Skew Vertex
                          15.980 [ 2:  ]  19% Sort
                          22.900 [ 2:  ]   8% Sum
------------------------------------------------------------------------------
       Data Bytes       (%)          Records       Status       Skew Flow Vertex    Port
       22,110,000     (55%)          221,100 [     :   2:     ]   0% f1   Sort      in
       44,220,000                    442,200 [     :   2:     ]   0% f2   Sum       in
------------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords "flows" "times" "spillage" "file-percentages"

Specifying **flows**, **times**, **spillage**, and **file-percentages** produces a report in which:

- Both flow and process CPU usage monitoring are turned on
- Extra columns are added to show:
  - Bytes Spilled for flow buffering
  - Maxcore usage and the amount of data spilled to disk
- A percentage-done (**%**) column is added to show the progress of file-input flows

Here is an example:

```
    % export AB_REPORT="flows times spillage file-percentages"
    % mp run
------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
        Maxcore  Spilled Bytes Records  CPU Time  Status Skew Vertex
                                          15.980 [ 2:  ]  19% Sort
            66%       3,000,000  30,000   22.900 [ 2:  ]   8% Sum
------------------------------------------------------------------------------
       Data Bytes   (%)/Buf          Records       Status       Skew Flow Vertex    Port
       22,110,000     (55%)          221,100 [     :   2:     ]   0% f1   Sort      in
       44,220,000       11M          442,200 [     :   2:     ]   0% f2   Sum       in
------------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

In the Flows (second) section of the report, note the "(%)/Buf" column. The column has alternative header labels because both **spillage** and **file-percentages** were specified for AB_REPORT. A single column is used to report these values, and the figures are distinguished as follows:

- If the flow is not buffered, the figure shows the amount of data consumed from the file (if any) from which the flow is taking data. This possibility is indicated by the column header label “(%)”. Consumption amounts are displayed in parentheses, with a “%” symbol appended.
- If the flow is buffered, the figure in this column shows the amount of data spilled to disk. This possibility is indicated by the column header label “Buf”. Spillage amounts are displayed with no parentheses.

Keywords “flows” “times” “spillage” “file-percentages” “totals”

Specifying **flows**, **times**, **spillage**, **file-percentages**, and **totals** produces a report combining all the information contained in the previous example, in addition to a Total line showing total CPU usage and spillage for all processes.

```
% export AB_REPORT="flows times spillage file-percentages totals"
% mp run
```

```
-------------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
        Maxcore  Spilled Bytes Records  CPU Time  Status Skew Vertex
                                          15.980 [ 2:  ]  19% Sort
            66%      3,000,000  30,000    22.900 [ 2:  ]   8% Sum
                     3,000,000  30,000    38.880 [ 4:  ]      Total
-------------------------------------------------------------------------------
       Data Bytes  (%)/Buf         Records      Status      Skew Flow Vertex   Port
       22,110,000    (55%)         221,100 [    :   2:    ]   0% f1   Sort     in
       44,220,000      11M         442,200 [    :   2:    ]   0% f2   Sum      in
-------------------------------------------------------------------------------
```

Note that with no report interval set (with **interval**), the report information is updated at the default rate, i.e. every second.

Keywords “times” “totals” “spillage” “split-cpu”

Specifying **times**, **totals**, **spillage**, and **split-cpu** produces a report in which process CPU usage time is shown, along with additional “Maxcore” and “Spilled Bytes/Records” columns, giving information on maximum core usage and the amount of data spilled to disk. **split-cpu** produces columns in which the CPU usage figures are given for system and user process activity separately.

```
% export AB_REPORT="times totals spillage split-cpu"
% mp run
```

```
---------------------------------------------------------------------------
Jan 11 07:00:30   Phase 0 running (30 seconds)
        Maxcore  Spilled Bytes Records  Usr CPU  Sys CPU  Status Skew Vertex
                                         13.580    2.400 [ 2:  ]  19% Sort
            66%      3,000,000  30,000   18.400    4.500 [ 2:  ]   8% Sum
                     3,000,000  30,000   31.980    6.900 [ 4:  ]      Total
---------------------------------------------------------------------------
```

Skew and its measurement

"Skew" is a measure of the relative imbalance of the partition sizes involved in parallel processing.

Ideally, all the partitions of a dataset or flow in a graph that employs parallelism are equal in size or nearly so. When this is true (and if all other conditions are equal), the work involved in processing the partitions is equally divided, and the efficiency of the parallelism is fully realized. Skew can thus be an indirect measurement of graph efficiency.

Reporting skew data

Flow and dataset skew for a graph are shown when you specify **flows** to AB_REPORT in the graph's environment.

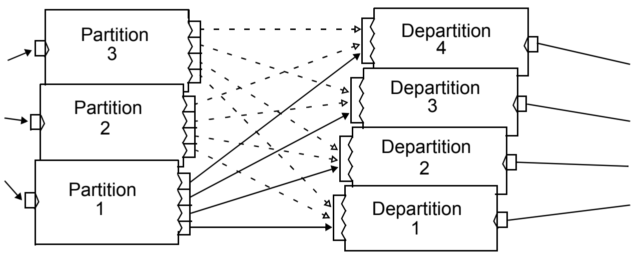
Specifying **skew** (in addition to **flows**) to AB_REPORT has the special effect of producing report data on the skew of the individual *plies* (see below for the definition of this term) of flows.

In the following sections, all types of skew measurement are discussed.

Flow plies

A flow partition is here called a *ply* to distinguish it from a data partition. There is a slight difference between the two. In cases of all-to-all flows, the number of flow partitions (or plies, as we call them in this discussion) will necessarily be greater than the number of partitions which they are connecting.

For example, consider this expanded view of a graph fragment in which data is moving from three to four partitions across an all-to-all flow:



Here there are 12 flow plies over which the data moves from the three to the four data partitions. It thus helps to avoid confusion to speak of data *partitions*, but flow *plies*.

Average multifile data partition size

In a perfectly balanced multifile, each data partition has the same size. That size is equal to the average partition size, which is the sum of all the partition sizes, divided by the number of partitions:

$$\text{average partition size} = \frac{\text{sum of sizes of all partitions}}{\text{number of partitions}}$$

Average size of a flow ply

The average size of a single flow ply is the sum of the sizes of all the plies, divided by the number of plies:

$$\text{average flow size} = \frac{\text{sum of sizes of all flow plies}}{\text{number of flow plies}}$$

Skew of a data partition

The skew of a data partition is the amount by which its size deviates from the average partition size, expressed as a percentage of the largest partition:

$$\text{skew of a partition} = \frac{\text{partition size} - \text{average partition size}}{\text{size of largest partition}} \times 100\%$$

The skew of a data partition is negative if it is smaller than the average size and positive if it is larger. The sum of all skews for a partition is 0%.

Skew of a flow

The skew of a flow is the amount by which its size deviates from the average flow ply size, expressed as a percentage of the largest flow ply:

$$\text{skew of a flow} = \frac{\text{flow size} - \text{average flow ply size}}{\text{size of largest flow ply}} \times 100\%$$

The skew of a flow ply is negative if it is smaller than the average and positive if it is larger. The sum of all ply skews for a flow is 0%.

Skew information for individual flow plies is shown if you specify **skew** and **flows** to AB_REPORT.

Showing flow ply skew with AB_REPORT

When you specify **skew** to AB_REPORT, individual extra lines are produced in the report under the report line for each flow. Each of these individual lines contains data about the skew of a single flow ply. By default, only four of the plies are show, but you can change this by specifying

skew=[*min_skew*[**:***max_nr_of_plies*]] where *max_nr_of_plies* is the maximum number of flow plies to show in the report (*min_skew* is the absolute value of a threshold skew value; all plies with skew greater than this value are to be shown).

You can specify

skew=[*min_skew*]**:all**

or

skew=[*min_skew*]**:***

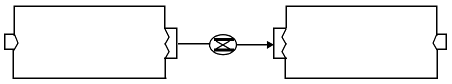
to show skew for all flow plies.

Each flow ply skew report line shows:

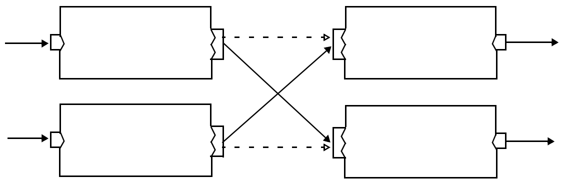
- The ply's relation to the average ply size for the flow (expressed as a percentage)
- The ply's skew, expressed as a negative or positive number (see "Skew of a flow" on page 68)

Insignificant skew

You should bear in mind that a high skew rate reported against an all-to-all flow is often not a sign of imbalanced processing. For example, consider the following graph fragment, in which data passes between two data partitions across an all-to-all flow:



As seen in an expanded view, the fragment might look like this:



Assume that the data coming in from the left is balanced between the two partitions. Likewise, the data is balanced coming out of the two partitions on the right.

However, let's also assume that during the processing that occurs between the components connected by the all-to-all flow, data passes over only the flow plies shown as solid lines, and none across the plies shown as dotted lines. As a result, 50% skew is reported against the all-to-all flow. Nevertheless, there is no imbalance in the way the partitions are handled; the "skew" is only a temporary side-phenomenon of the processing.

Thus, you should always check the downstream skews in a graph to make sure that skews reported against all-to-all flows are significant.

NOTE: If you use the AB_MAX_DATA_PARALLELISM configuration variable, you should ignore the reported skew values. The mechanism limits parallelism by inserting phase breaks in the graph, which gives inaccurate skew readings.

Skew of a multifile

The skew of a multifile is the amount by which the performance of the multifile suffers as a result of imbalance among its partitions.

The skew of a multifile is equal to the skew of the partition with the largest positive value. An overall skew of 34%, for example, means that skew is causing a performance penalty of 34% relative to a perfectly balanced multifile. Partitions with high positive values are processing more data than they need to and are the ones causing the graph to run unnecessarily slowly.

The following examples are an analysis of the skews of two multifiles, each with four partitions.

Example: A highly skewed multifile

A multifile system with 100 MB has four partitions of the following sizes: 10 MB, 20 MB, 30 MB, and 40 MB.

The average size of a partition is 25 MB:

```
(10 MB + 20 MB + 30 MB + 40 MB) / 4 = 25 MB
```

This is the size each partition would need to have in order to result in no skew—in other words, perfectly balanced data across all four partitions.

The skews of each of the partitions are as follows:

$$\text{skew of 10 MB partition} = \frac{10-25}{40} = \frac{-15}{40} = -37.5\,\%$$

$$\text{skew of 20 MB partition} = \frac{20-25}{40} = \frac{-5}{40} = -12.5\,\%$$

$$\text{skew of 30 MB partition} = \frac{30-25}{40} = \frac{5}{40} = 12.5\%$$

$$\text{skew of 40 MB partition} = \frac{40-25}{40} = \frac{15}{40} = 37.5\,\%$$

The skew of the multifile is 37.5%, which is the skew of the partition with the largest positive value.

Example: An extremely skewed multifile

A multifile system with 100 GB has four partitions of the following sizes: 0GB, 0 GB, 0 GB, and 100 GB.

The average size of the multifile partitions is 25 GB:

```
(0 GB + 0 GB + 0 GB + 100 GB) / 4 = 25 GB
```

This is the size each partition would need to have in order to result in no skew, or perfectly balanced data across all the partitions.

The skew of each of the partitions is as follows:

$$\text{skew of first 0 GB partition} = \frac{0-25}{100} = \frac{-25}{100} = -25\%$$

$$\text{skew of second 0 GB partition} = \frac{0-25}{100} = \frac{-25}{100} = -25\%$$

$$\text{skew of third 0 GB partition} = \frac{0-25}{100} = \frac{-25}{100} = -25\%$$

$$\text{skew of 100 GB partition} = \frac{100-25}{100} = \frac{75}{100} = 75\%$$

The skew of the multifile is 75%, the skew of the partition with the largest positive value.

Troubleshooting skew

The best way to troubleshoot high skew is to track it back to its source. Situations that might lead to high skew are unbalanced data, an overloaded computer, or different computer speeds.

A typical highly skewed situation occurs in a Rollup component after a partitioner has divided a single data flow into several very unbalanced data flows.

Example of a high-skew situation

A Partition by Key component partitions data based on zip code. The partitions are unbalanced because the distriibution of the zip code values themselves is unbalanced.

To reduce the skew and improve performance, you might consider the following approaches:

- You could replace the Partition by Key based on zip codes with a Partition by Round-robin component. This will ensure evenly-balanced partitions.
- Add a Rollup component, rolling up the data records by zip code within each partition.
- Add either a Partition by Key or a Merge component.

  Use a Partition by Key component (using the zip code as key) if you want to perform further processing in parallel.

  use a Merge component if you want to perform further processing without parallelism.
- Perform a second rollup of the data records among all the partitions.

Another highly skewed situation can occur when using the Input Table component. The Co>Operating System partitions data records using the row numbers of the first and last record from the table being unloaded, and this can result in an uneven distribution of data records among the specified number of partitions.

To reduce the skew and improve performance, you can try specifying more partitions in the unload than you ultimately want to use, in order to increase the chances of making the partitions more even. For example, you could unload the database data using 16 partitions, then repartition the data from 16 to 4 partitions using the Partition by Round-robin component.

SDE utilities for reporting skew

In the SDE, indicators of partition skew are produced by the Co>Operating System utilities **m_df**, **m_du**, and **m_ls**.

Summary reports

The format of summary files is defined in the record format file **$AB_HOME/include/summary.dml**. For example, when applied to the contents shown above, the **summary.dml** record format yields the following:

```
Record 1:

[record
    kind         "job-start"
    name         NULL
    partition    NULL
    state        NULL
    num_records  NULL
    num_bytes    NULL
    cpu_time     NULL
    elapsed_time "0"
    dt           "1998-11-24 "
    hh           "15"
    mm           "03"
    ss           "22"]
Record 2:

[record
    kind         "component"
    name         "sel"
    partition    "0"
    state        "finished"
    num_records  NULL
    num_bytes    NULL
    cpu_time     "0.430"
    elapsed_time NULL
    dt           NULL
    hh           NULL
    mm           NULL
    ss           NULL]
```

The contents of **$AB_HOME/include/summary.dml** are as follows:

```
record
    string(" ") kind;
    if (kind == "flow" || kind == "component")
        begin
            string(" ") name;
            decimal(" ") partition;
            string(" ") state;
        end;
    if (kind == "flow")
        begin
            decimal(" ") num_records;
            decimal("\n") num_bytes;
        end;
    if (kind == "component")
        decimal("\n") cpu_time;
    if (kind == "job-start"
            || kind == "phase-end")
        begin
            decimal(" ") elapsed_time;
            date("YYYY-MM-DD ") dt;
            decimal(":") hh, mm;
            decimal("\n") ss;
        end;
end
```

Summary record format description

The meanings of the fields are as follows.

- **kind**

  The **kind** field determines the interpretation of the rest of the record, as follows:

flow	Flow partition
component	Component partition
job-start	Start of job
phase-end	End of phase

- **name**

  For components, the **name** field holds the name of the component. For flows, the **name** field holds the name of a port at one end of the flow in the form *component-name.port-name*.

- **partition**

  The **partition** field holds the partition number.

- **state**

  The **state** field holds an indication of the state:

  For flows, the **state** field may take on these values:

unopened	Flow was never opened
opened	Flow was opened
closed	Flow was opened, drained, and closed

For components, the **state** field may take on these values:

unstarted	Program was never started
started	Program was started
finished	Program finished successfully
error	Program terminated in error state
debug	Program was being debugged

- **cpu_time**

  The **cpu_time** field holds the CPU time consumed by a component, in seconds.

- **num_records**

  The **num_records** field holds the number of records carried by a flow.

- **num_bytes**

  The **num_bytes** field holds the number of bytes carried by a flow.

- **elapsed_time**

  The **elapsed_time** field holds the number of seconds elapsed since the start of the phase.

- **dt**, **hh**, **mm**, and **ss**

  The **dt**, **hh**, **mm**, and **ss** fields hold a timestamp for job start and phase ends.

Using the summary file contents

Because it is a fully DML-described dataset of records, the summary file can be processed by graphs which produce reports for auditing or performance analysis purposes.

For example, the following transform function, used in an Aggregate component with the null key specifier ({}), sums up CPU time from all partitions of all components and elapsed time from all phases:

```
out :: add_up_times(agg, in) =
begin
        out.cpu_time ::
            (if (is_defined(agg)) agg.cpu_time else 0) +
            (if (is_defined(in.cpu_time)) in.cpu_time else 0);
        out.elapsed_time ::
            (if (is_defined(agg)) agg.elapsed_time else 0) +
            (if (is_defined(in.elapsed_time)) in.elapsed_time
else 0);
end
```

Processing the example summary dataset shown at the beginning of this section (page 61) with the above transform would yield the following record:

```
[record
   elapsed_time "6"
   cpu_time     "8.38"]
```

The example in **$AB_HOME/examples/dml/transforms/summary/** shows a graph that processes a summary dataset to produce a report.

Display modes

You can display the monitoring output of deployed graphs in one of two modes at the command line:

- Scrolling mode, which prints reports one after another.
- Display mode, which prints a report by overwriting the previous one.

Display modes work only for windows and terminals that use ANSI cursor-control conventions. These include most windows and terminals. Display mode is turned off if the TERM configuration variable is set to **emacs**, **dumb**, or **tty**, or if you specify **scroll=true** in AB_REPORT.

You should manually specify **scroll=true** if you see something like this:

```
^[[A^[[A^[[A^[[J
```

Alternatively, you can redirect monitor output to a file or to a named pipe by specifying **file=***filename* in AB_REPORT or **mp run -report**. By default, this appends reports to the file specified. The file is not truncated before the job starts. Reports are terminated by a single line containing an ASCII form-feed character (**^L**). However, specifying **scroll=false** causes the file to overwrite the previous one.

Setting **scroll=false** does not work with named pipes, since the reporting mechanism recreates the target file with each report.

Using mp run -report

The amount of reporting done by a graph deployed from the GDE as a script can also be controlled by specifying various **-report** arguments (these are mostly identical to the AB_REPORT settings) to the **mp run** command in the deployed script itself.

Any AB_REPORT setting can be specified to **mp run -report**, except for **monitor** and **process**.

Each setting specified must be preceded by a separate **-report** flag. For example, the following command

```
mp run -report flows -report interval=60 -report times -report scroll=true
```

produces a series of listings similar to those produced by the default AB_REPORT setting described earlier.

You can get a listing of all the valid **-report** arguments (and much else) by typing

```
mp -help
```

on a shell command line.

NOTE: You should use AB_REPORT (rather than **mp run -report**) to specify report parameters; use of the **-report** option to **mp run** is deprecated.

5>

Addendum

This chapter contains information additional to the contents of the preceding chapters.

m_cleanup, m_cleanup_rm, m_cleanup_du utilities

The Ab Initio cleanup utilities perform the following tasks:

- find any Ab Initio temporary files and directories on a machine
- display the amount of disk space they use
- remove them

The execution of a cleanup utility affects the temporary files and directories only on the machine on which it executes. To clean up files and directories on other machines, you must execute the utility on each of the other machines individually.

The utilities do not affect recovery (**.rec**) files. The cleanup utilities do not list, remove, or display the very slight amount of disk space used by recovery files themselves. The utilities work independent of recovery files and do not require recovery files to be present. Once temporary files and directories of a graph are gone, however, any attempt to rollback the graph fails.

For a discussion of recovery from failed jobs–lacking material about the the new cleanup utilities–see pages 58-61 of the *Co>Operating System Administrator's Guide*.

The cleanup utilities are:

m_cleanup (page 81)	**m_cleanup.exe** on Windows
m_cleanup_rm (page 84)	**m_cleanup_rm.bat** on Windows
m_cleanup_du (page 84)	**m_cleanup_du.bat** on Windows

One way to use the cleanup utilities is to use **crontab**, or another scheduling program, to schedule periodic cleanups of temporary files and directories. For example, the following command, run by superuser nightly, removes temporary files and directories associated with jobs older than 7 days:

```
find /var/abinitio/ -name '*.[hn]lg' -mtime +7 | xargs m_cleanup_rm -j
```

Background

The Co>Operating System automatically creates a recovery (**.rec**) file and other temporary files and directories in the course of executing a graph. When a graph terminates abnormally, it leaves the temporary files and directories on disk. At this point there are several alternatives possible:

- Roll back to the last checkpoint.

The Co>Operating System rolls back the graph automatically, if possible; you can rollback the graph manually by explicitly using the **m_rollback** command without the **-d** argument.

After a rollback, some temporary files and directories remain on disk. To remove them, follow one of the other three alternatives.

- Rerun the graph.

  If the graph is not already rolled back, rerunning the graph first rolls back the graph to last checkpoint. The graph then starts re-executing. If the re-execution is successful, it removes all temporary files and directories.

- Roll back and clean up using **m_rollback -d**.
- Clean up using the **m_cleanup** utilities.

Difference from m_rollback -d

There are several important differences between **m_rollback -d** and **m_cleanup_rm**.

- For jobs that run on more than one computer, **m_rollback -d** operates on all the computers at once; but the **m_cleanup** utilities operate on only one computer at a time.
- The **m_cleanup_rm** utility has facilities for finding and cleaning up old jobs, whereas **m_rollback -d** needs a recovery file to operate on.
- The **m_rollback -d** utility restores the previous contents of Output Files, whereas **m_cleanup** does not.

The second and third differences are what make **m_cleanup_rm** much more suited for cleaning up old jobs.

m_cleanup

The **m_cleanup** utility lists temporary files and directories.

Syntax

The utility has the following forms:

m_cleanup
m_cleanup -help
m_cleanup -j *job_log_file* **[** *job_log_file* ... **]**

Arguments

None.	Prints usage for the command.
-help	Prints usage for the command.

-j *job_log_file*	Lists the temporary files and directories listed in the log file specified by *job_log_file*. To specify multiple log files, separate each file with a space. The *job_log_file* can be an absolute or relative path name. Log files have either a **.hlg** or **.nlg** suffix. A log file ending in **.hlg** is on the host. A log file ending in **.vlg** is on a processing, or vnode, computer of a graph. Paths to log files have the following syntax: • On the host: *AB_WORK_DIR*/**host**/*job_id*/*job_id*.hlg • On a processing computer: *AB_WORK_DIR*/**vnode**/*job_id*-*XXX*/*job_id*.**nlg** The *XXX* on the processing computer path is an internal ID assigned to each computer by the Co>Operating System

Details

The **m_cleanup** utility lists temporary files and directories in the order they are listed in the graph log files. The utility does not use separators to distinguish the file-and-directory list of one log file from that of others.

The utility affects the files, directories, and log files only on the machine that is running the utility. To clean up files and directories on other machines, you must execute the utility on each of the other machines individually. A executing graph creates a log file on the host and on each of its processing computers.

The utility runs safely while graphs are executing. It does not affect their temporary files and directories.

The utility fails to identify temporary files and directories if the graph executed while the rarely used AB_LAYOUT_EXACT configuration variable is set to true.

The utility relies on shared memory to read the log files and is prone to shared memory problems. For instance, if a graph executes with AB_SHMEM_BYTES set very high when **m_cleanup** executes with a lower value, errors like the following two may appear:

```
Heap initialized with too small an area
   or too small a quantum
Aborting
Shared memory heap ran out of space.
Re-run with a larger value of AB_SHMEM_BYTES,
   which is currently 10000.
Memory fault (core dumped)
```

If you see errors like these, increase the value of AB_SHMEM_BYTES to allow proper loading of the log file and then rerun the utility. For example:

```
export AB_SHMEM_BYTES=1000000000
m_cleanup
```

Examples

The following example uses the **find** command to search the directory **/var/abinitio** for log files whose owner is **jdoe** and whose modification date is more than two days ago. The command pipes the output to **xargs**, which buffers the output for input to **m_cleanup**. The **m_cleanup** utility lists the temporary files and directories listed in the log files. For information about **find** and **xargs**, see the UNIX manual page.

```
find /var/abinitio/ -name "*.[hn]lg" \
-user jdoe -mtime +2 | \
xargs m_cleanup -j
```

The following example uses the **find** command to search the directory **/var/abinitio/** for log files whose owner is **jdoe** and whose modification date is more than eight days ago. The command pipes the output to **xargs**, which buffers the output for input to **m_cleanup**. The **m_cleanup** utility lists the temporary files and directories in the log files and pipes the output to **xargs**, which buffers the output for the **rm** command. The **-r** switch of **rm** removes subdirectories and their contents recursively. The **-f** switch removes items without confirmation. For information about **xargs** and **rm**, see their UNIX manual pages.

```
find /var/abinitio/ -name "*.[hn]lg" \
-user jdoe -mtime +8 | \
   xargs m_cleanup -j | \
   xargs rm -rf
```

Once temporary files and directories of a graph are gone, any attempt to rollback the graph fails. For example, the following series of commands removes all temporary files and then attempts to rollback the graph. The result is an error message.

```
find /var/abinitio/ \
        -name "*.[hn]lg" -mtime +7 |
        xargs m_cleanup -j | xargs
rm -rf
m_rollback -d my_failed_graph.rec
ABINITIO: Aborted
ABINITIO: Auto rollback failed
```

m_cleanup_rm

The **m_cleanup_rm** utility removes temporary files and directories.

Syntax

The **m_cleanup_rm** utility has the same syntax as **m_cleanup**.

Arguments

The arguments of **m_cleanup_rm** have the same meanings as those of **m_cleanup**.

Details

The **m_cleanup_rm** utility:

- Applies its arguments to **m_cleanup**, which produces as output the list of temporary files and directories to remove.
- Pipes the output to the **rm** utility, which removes the files and directories.

Examples

The following example uses **m_cleanup_rm** to remove the temporary files and directories of user **jdoe** for jobs over seven days old.

```
find /var/abinitio/ -name "*.[hn]lg" \
    -user jdoe -mtime +7 | xargs m_cleanup_rm -j
```

m_cleanup_du

The **m_cleanup_du** utility displays the amount of disk space used by temporary files and directories

Syntax

The **m_cleanup_du** utility has the same syntax as **m_cleanup**.

Arguments

The arguments of **m_cleanup_du** have the same meanings as those of **m_cleanup** except that the *job_log_file* argument must be an absolute path name, because **m_cleanup_du** executes in a temporary directory.

For example, the argument **-j ./a_job_id.hlg** *works* with the following **m_cleanup** command:

```
cd /var/abinitio/host/a_job_id
m_cleanup -j ./a_job_id.hlg        (Works)
```

However, the argument *does not work* with the following **m_cleanup_du** command:

```
cd /var/abinitio/host/a_job_id
m_cleanup_du -j ./a_job_id.hlg      (Does not work)
```

The following **m_cleanup_du** command works, because it specifies an absolute pathname:

```
m_cleanup_du -j \
/var/abinitio/host/a_job_id/a_job_id.hlg (Works)
```

Details

The **m_cleanup_du** utility first applies its arguments to **m_cleanup**, thereby identifying the temporary files and directories whose disk space it is to display.

The **m_cleanup_du** utility displays a single number representing the number of KB used by the temporary files and directories listed in the specified log files.

Do not embed **m_cleanup_du** in a graph. Unlike **m_cleanup** and **m_cleanup_rm**, the implementation of **m_cleanup_du** itself is as a graph.

Examples

The following example uses **m_cleanup_du** to display the disk space used by the temporary files and directories of the user **jdoe** that are over seven days old. The resulting number, **140**, shows that using **m_cleanup_rm** with the same arguments would free 140 KB of disk space.

```
find /var/abinitio/ -name "*.[hn]lg" -user jdoe \
    -mtime +7 | xargs
m_cleanup_du -j
    140K
```

m_env

m_env is a general purpose utility for obtaining information about Ab Initio configuration variable settings in an environment.

Following is a list of all **m_env** arguments:

No arguments	Displays the values of major Ab Initio configuration variables.
-alias *host1* [*host2* ...]	Displays the alias for hosts *host1* [*host2* ...]. If you specify more than one hostname, separate the names with spaces.
-D	Describes the Ab Initio documentation.
-a	Displays all configuration variables in the environment.
-all	Same as **-a**.
-compatibility	Displays configuration variables that turn on backward compatibility modes
-d	Displays documentation about debugging.
-debug	Displays configuration variables that provide debugging assistance.
-debug-info	Same as **-d**.
-describe	Displays explanations of some Ab Initio configuration variables.
-doc	Same as **-D**.
-environment	Displays configuration variables that describe the computing environment.
-error	Displays configuration variables that control error suppression and handling.
-f[**ind**] *string*	Causes **m_env** to print configuration variables whose names or documentation contains a specified string. See "Use of the -find argument" on page 87.
-g *varname* [*computer* | *alias*]	Displays the value of the configuration variable *varname* on *computer* or computer *alias*. If you omit *computer* and *alias*, **m_env** uses the local host. If *varname* is not set, returns **<unset>**.
-get *varname* [*computer* | *alias*]	Same as **-g**.

-get-value *varname* [*computer* | *alias*]	Same as **-g**.
-gs *varname* [*computer* | *alias*]	Same as **-g** except that, if *varname* is not set, returns nothing, that is, an empty string.
-get-silent *varname* [*computer* | *alias*]	Same as **-gs**.
-graph	Displays configuration variables that affect graph processing.
-h	Displays explanations of some Ab Initio configuration variables.
-help	Explains each **m_env** shell argument
-map	Displays a mapping of previous XX_* and current AB_* configuration variable names.
-monitoring	Displays configuration variables that control monitoring.
-n {*computer* | *alias*}	Displays the configuration variable settings for *computer* or computer *alias*.
-computer-info {*computer* | *alias*}	Same as **-n**.
-obsolete	Displays explanations of obsolete configuration variables.
-startup	Displays configuration variables that are used in attaching to remote computers.
-tuning	Displays configuration variables that tune operating system and communication performance.
-v	Displays Ab Initio release information.
-version	Same as **-v**.
-w or **-wizard**	Displays explanations of Ab Initio configuration variables for expert users.

The arguments **-a** and **-all** cause **m_env** to display all the configuration variables available in the environment.

Use of the -alias argument

The argument **-alias**, which resolves host aliases and displays them, is useful for resolving host aliases in shell scripts.

Use of the -find argument

The **-find** or **-f** argument to **m_env** causes **m_env** to print configuration variables whose names or documentation contains a specified string. The syntax is:

-f[**ind**] *string*

If the case of *string* is:

Uppercase	**m_env** searches the names of configuration variables.
Lowercase or mixed case	**m_env** searches the names of configuration variables as well as their documentation. The search is case insensitive.

The **m_env** utility can use **-find** with **-wizard**, **-obsolete**, and **-describe**. With the **-wizard** argument, **-find** looks at configuration variables used by advanced users. With **-obsolete**, **-find** looks at obsolete configuration variables. With **-describe**, the output is verbose.

For example, the following command:

```
$ m_env -find PASS
```

Prints the following:

```
Found by name:

Current Configuration Variable Settings for Ab Initio:
    (use "m_env -describe" for descriptions)

Variable               Set  Value or Resulting Action
--------------------   --- ----------------------------------
AB_IDB_PASSWORD             <unset>
AB_PASSPHRASE               <unset>
AB_PASSWORD              *  "abc123" (from /u/joe/.netrc)
AB_RTEL_PASSWORD_PROMPT       "Password:|password:"
```

The following command:

```
$ m_env -find pass
```

Prints the following:

```
Found by name:

Current Configuration Variable Settings for Ab Initio:
          (use "m_env -describe" for descriptions)

Variable                Set  Value or Resulting Action
--------------------    ---  ---------------------------------
AB_IDB_PASSWORD              <unset>
AB_PASSPHRASE                <unset>
AB_PASSWORD              *   "abc123" (from /u/joe/.netrc)
AB_RTEL_PASSWORD_PROMPT        "Password:|password:"

Found by description:

Current Configuration Variable Settings for Ab Initio:
          (use "m_env -describe" for descriptions)

Variable                Set  Value or Resulting Action
--------------------    ---  ---------------------------------
AB_AIR_PROJECT               <unset>
AB_LOCAL_NETRC               /u/joe/.netrc
```

Individual configuration variables

Individual configuration variables may also be specified.

m_kill

The **m_kill** utility kills a running job. The command blocks until it can verify that the job is killed or after AB_TIMEOUT seconds, whichever comes first. The default for AB_TIMEOUT is **30** seconds.

Following is the syntax of **m_kill**:

m_kill [*signal*] {*jobname*.**rec** | *jobname* }

signal	The *signal* argument is optional. If you use it, it must have one of the following values: • **-TERM** Kills the job and triggers a rollback. This is the default. • **-KILL** Kills the job immediately without triggering a rollback. You must issue the command **m_rollback** later. • **-QUIT** Kills the job immediately without triggering a rollback and forces the **Xmp run** process to dump core (system limits permitting).
jobname.**rec**	Name of a recovery file for the job you want to kill, for example, **my-job.rec**. You must use either *jobname*.**rec** or *jobname*.
jobname	Name of the job you want to kill, as given in the **mp job** command. You must use either *jobname*.**rec** or *jobname*.

m_mkfile

The **m_mkfile** utility creates an "explicit multifile" out of an arbitrary set of files. The utility creates a control partition and associates it with the specified files, which thereby become the multifile's data partitions.

Syntax

The following form creates an explicit multifile system by naming the URLs of the files (which become the data partitions of the multifile) on the command line.

m_mkfile [**-newfiles**] [**-m**[**ode**] *mode*] [**-f**] [**-rmdata**] \
control_url partition_url [*partition_url* ...]

The following form creates an explicit multifile system by naming the files from a list of their URLs piped from standard output.

m_mkfile [**-newfiles**] [**-m**[**ode**] *mode*] [**-f**] [**-rmdata**] \
control_url **-**

The following form creates an explicit multifile system by naming the files by reading their URLs from a file. Newlines or whitespace must separate each file named in the file.

m_mkfile [**-newfiles**] [**-m**[**ode**] *mode*] [**-f**] [**-rmdata**] \
control_url **-f** *url*

Arguments

[No arguments]	Prints the usage statement for the utility.
-newfiles	Generates a uniquely named file for each *partition_url*. • If *partition_url* is a file, **m_mkfile** creates the new file in the same directory. • If *partition_url* is a directory, **m_mkfile** creates a new file in that directory.

	If you supply only one *partition_url*, and it is: • A multidirectory, **m_mkfile** creates a new file in each of the data partition directories of that multidirectory. • An existent or non-existent multifile, **m_mkfile** creates a new file in each of the data partition directories of that multifile. • Only a host name and the configuration variable AB_EXPLICIT_MULTIFILE_DEFAULT_DIR is defined for that host, **m_mkfile** creates the new file on the host in the directory specified by that configuration variable. • Only a host name and the configuration variable AB_EXPLICIT_MULTIFILE_DEFAULT_DIR is not set, then **m_mkfile** creates a subdirectory named **explicit** in the directory AB_WORK_DIR. The configuration variable is also used if the directory for the new file does not exist or cannot be written to. The new file names have the form *control_url_base*-*unique_string*. The *control_url_base* is the path to *control_url*.
-m[**ode**] *mode*	Sets the permissions for *control_url* and *partition_url*, if they do not already exist. The *mode* behaves in the same way as the mode argument of the UNIX **chmod** command.
-f	Forces the recreation of *control_url* if it already exists. Without this or the **-rmdata** argument, the command will fail if *control_url* exists.

-rmdata	Removes data partitions, if any, referred to by *control_url*, if it exists before creating *control_url*. This argument implies **-f** if *control_url* exists. Without **-rmdata**, data partitions referred to by *control_url* are left intact.
control_url	Specifies the URL of the control partition of the explicit multifile system. The URL must not be in a multifile system.
partition_url	Specifies the URL of one of the arbitrary data partitions.

Examples

The following example creates an explicit multifile **x** that refers to existing data files **a.dat** and **b.dat**:

```
m_mkfile x //pluto.us.com/p/dir/a.dat \
    //apple.us.com/p/dir/b.dat
```

The following example creates an explicit multifile **x** that refers to two newly created files, one in directory **dir-1** and the other in directory **dir-2** (that is, in the same directory that contains file **a.dat**.

```
m_mkfile -newfiles x //pluto.us.com/p/dir-1 \
    //apple.us.com/dir-2/a.dat
```

In addition, the utilities **m_rm** and **m_mv** have a new argument **-rmdata**. With **m_rm**, the new argument removes the data partitions of an explicit multifile system:

-rmdata	Removes the data partitions of an explicit multifile system specified by **m_rm**. Without **-rmdata**, **m_rm** leaves the data partitions of the explicit multifile system intact. An explicit multifile system is one that has been created with **m_mkfile**.

With **m_mv**, the argument moves them:

-rmdata	Allows the move of an explicit multifile system. Without **-rmdata**, the **m_mv** command fails with an error message. Requiring this argument for moving an explicit multifile system safeguards against unintentionally removing the data partitions of an explicit multifile.

m_shminfo

The **m_shminfo** utility helps in the identification of orphaned memory segments.

As a result of an abnormal termination, the Co>Operating System may leave behind a shared memory segment with no processes attached to it. Given the identifier (**shmid**) of a shared memory segment, **m_shminfo** provides various information useful in determining whether the segment really is an orphan.

The command **ipcs -m** yields a list of all shared memory segments accessible by the current user, in a format that varies slightly by platform. One of the fields displayed for each segment is its unique identifier, marked **shmid** or **ID** in the column heading.

Consider this result of issuing **ipcs -m** on a Linux system:

```
paiute-l$ ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00280267 1          root       644        1024000    1
0x00000000 6920706    ephraim    600        10000008   0
```

The result indicates a stray segment **6920706**:

Consider this result of issuing **m_shminfo 6920706**.

```
paiute-l$ m_shminfo  6920706
6920706  1b14320a-3c43a895-5180  paiute-l.abinitio.com  Mon Jan 14 \
          22:57:09 2002  /tmp/ephraim/abworkdir
```

Running **m_shminfo** with the identifier **6920706** displays the Co>Operating system virtual machine ID, originating system, starting date and time, and AB_WORK_DIR. This information makes it possible to locate and identify the job that gave rise to this shared memory segment, and decide whether the job is still active or needs cleanup.

The following is the syntax of the **m_shminfo** command.

m_shminfo *shmid* ...

The **m_shminfo -help** command gives a terse summary of the usage and the output.

m_view_errors

The command-line utility **m_view_errors** translates machine-readable error information into text format. The syntax of **m_view_errors** is the following:

> **m_view_errors** [**-help**] [**-quiet**] [**-terse**] [**-verbose**]
> [**-errnum**] [**-noerrnum**] [**-read_until** *string*]

The following are the arguments of **m_view_errors**:

-help	Prints the usage statement for the utility.
-quiet	Suppresses warnings.
-terse	Shows only basic error messages, corresponding to the usual text-based error messages (DEFAULT).
-verbose	Shows full, verbose details on each error.
-errnum	Shows error numbers (DEFAULT).
-noerrnum	Hides error numbers.
-read_until *string*	Continues reading input until the utility encounters *string*.

A new configuration variable AB_STRUCTURED_ERRORS controls the type of format in which the Co>Operating System emits errors.

- If AB_STRUCTURED_ERRORS is undefined, the Co>Operating System emits all error messages in text format.
- If AB_STRUCTURED_ERRORS=1, the Co>Operating System emits context-specific error messages in machine-readable format.

  To read error messages generated in machine-readable format invoke the command-line utility **m_view_errors**.

m_wc

The **m_wc** utility counts records and bytes in one or more data files.

Following is the syntax of **m_wc**:

m_wc [**-no-commas**] *metadata data*

-no-commas	Suppresses commas from output. Without this argument, the output includes commas in order to make the numbers more readable.
metadata	One of the following: *url* — The URL of the file containing the record format of *data*. **-string** *string* — The record format of *data*.
data	One of the following: *url* ... — One or multiple URLs, each one of which represents a file for which **m_wc** is to count records and bytes. **-string** *string* ... — One or multiple **string** arguments, each one of which is the data for which **m_wc** is to count records and bytes.

The **m_wc** command returns one line for each *data* argument. Each line contains:

- The record count
- The byte count
- The filename of *data* if supplied

The following are examples.

```
$ m_wc -string 'void(1)' mfs/mydata.dat
       4,297,064,448          4,297,064,448 mfs/mydata.dat
$ m_wc myrecfmt.dml mfs/mydata.dat
            209,7152          4,297,064,448 mfs/mydata.dat
```

The mp command

The following are features related to the Co>Operating System graph command, **mp**.

mp *flowtype*

You can set permissions on flows through named pipes and files by using the **-mode** argument with **mp** *flowtype*. You set the permissions either by specifying them exactly or by specifying changes to the default permissions. You cannot set permissions on TCP flows.

Use the following syntax to set permissions on a flow through a file or a named pipe.

> **mp** *flowtype flowname from-port to-port* ⇒
> **-mode** *modespec other_arguments*

The **flowtype** specifies one of the following types of flows (described in the *Co>Operating System Administrator’s Guide*):

- **all-to-all-flow**
- **fan-in-flow**
- **fan-out-flow**
- **straight-flow**

The **-mode** *modespec* argument of **mp** *flowtype* can set permissions numerically or symbolically, like the UNIX command **chmod**. For example, if you want the permissions to be **rw-rw-rw-**, use one of the following formats:

- In the numeric format:

```
0666
```

- In a symbolic format (either of the following):

```
ugo+rw
ugo=rw
```

The most likely value for the *modespec* is **g+rw**, to allow interaction with other group members. The default mode is **0600**, user-read-write only. Do not set permissions to be more restrictive that the default. If you do, the next job will fail.

mp *file_component*

The new features of **mp** *file_component* are:

- Exempting files from commit-rollback processing

You can exempt output files and temporary files from commit-rollback processing by using the **norollback** flag with **mp ofile, mp iofile**, or **mp file**. For example:

```
mp file out out.dat -flags \
   create,unlink,norollback
```

- Synonyms for **creat** and **trunc** flags

  We added the following arguments to **-flag** in **mp ofile** and **mp iofile**. The arguments are synonyms for existing, cryptic flag names.

create	Synonym for the **creat** flag.
truncate	Synonym for the **trunc** flag.

Syntax

The following is the complete syntax for **mp** *file_component.*

mp *file_component file_component_name* ⇒
file_url [*file_url* ...] ⇒
[**-flags** *flag* [*flag* ...]] [**-mode** *mode*]

Arguments

Following are the arguments to the **layout** command:

file_component	Specifies the type of file component. The *file_component* must be one of the following values: **ifile** Specifies an input file. Implies the flag rdonly. **ofile** Specifies an output file. Implies the flags **wronly**, **creat**, and **unlink**. **iofile** Specifies a temporary storage file. Implies the flags **rdwr**, **creat**, and **unlink**.
file_component_name	Specifies the name you want to give to this instance of the file component.

file_url	Specifies a file in the layout of the file component in URL or UNC format. If the layout is: • One serial file, use only one *file_url*. • A multifile, use only one *file_url*. • Multiple serial files, use a *file_url* for each serial file An **ifile** *file_url* must be readable. An **ofile** *file_url* must be writable. An **iofile** *file_url* must be both.
-flags *flag* [**,** *flag* ...]	Specifies access methods and special handling of the file. Separate flags with commas. Do not use spaces. If you do not use **rdonly**, **wronly**, or **rdwr**, the default is **rdonly**. The following are the possible values of *flag*: **append** When writing, appends output to end of the file if the file already exists. **creat** When writing, create the file if the file does not already exist. **create** Same as **creat**. **destroy** Deletes the file after job execution. **excl** Excludes other processes from using the file during job execution. **Enorollback** Exempts **ofile**s and **iofile**s from commit-rollback processing. **rdonly** Reads the file only. Do not use with **wronly** or **rdwr**. **rdwr** Writes the file and then reads it. Do not use with **rdonly** or **wronly**. **trunc** When reading, truncates the file forward. **truncate** Same as **trunc**. **unlink** Before writing, deletes the file if the file already exists. **wronly** Writes the file only. Do not use with **rdonly** or **rdwr**.

-mode *mode*	Specifies the permissions of files written during job execution. The *mode* can be numeric or symbolic. For example, if you want the permissions to be **rw-rw-rw-**, use one of the following formats:
	• Numeric format:
	`0666`
	• In a symbolic format (either of the following):
	`ugo+rw`
	`ugo=rw`
	The most likely value for the *mode* is **g+rw**, to allow interaction with other group members.The default mode is **0600**, user-read-write only.
	Do not set permissions to be more restrictive than the default. If you do, the next job will fail. Do not use with **rdonly**.

mp layout

The following arguments are new to **mp layout**:

-exact *urls*	Specifies that the component work directories are exactly as specified in the custom layout.
-workdir *urls*	Specifies that the component work directories are subdirectories of **.WORK**, new directories that Ab Initio creates as necessary beneath those specified in the custom layout. If you use the directory / or the name of a host, then **-workdir** specifies that the component work directories are subdirectories of $AB_WORK_DIR just like the hosts layouts. The default is **-workdir**.

The *urls* can be either directories or files. If *urls* are files, Ab Initio ignores the file and uses only the host and directory parts of the **urls**. The directories must be writable.

Previously, the layout implied by a file component consisting of one or more serial files consisted of directories under $AB_WORK_DIR on the files' hosts. Now it is the same as a custom layout naming these files.

If you want Ab Initio to behave as in previous releases, set both of the following configuration variables to **true**:

AB_LAYOUT_WORK_DIR	If true, layouts implied by serial files are under the $AB_WORK_DIRs on the files' hosts.
AB_LAYOUT_EXACT	If true, **-exact** mode is the default for custom layouts.

Syntax

The following is the complete syntax for the layout command:

mp layout *layout_name*
{ *url_unc* [*url_unc* ...] |
-hosts *computer_url_unc* [*computer_url_unc* ...] |
-exact *exact_url_unc* [*exact_url_unc* ...] |
-workdir *workdir_url_unc* [*workdir_url_unc* ...] }

Arguments

The following are the arguments to the **layout** command.

layout_name	Specifies the name you want to give to the layout.
url_unc [*url_unc* ...]	Same as **-workdir** unless the configuration variable AB_LAYOUT_EXACT is set to true. If AB_LAYOUT_EXACT is set to true, **mp layout** behaves as if you had used **-exact**. If you specify *url_unc* arguments that specify only computer names, the argument behaves as if you had used **-hosts** regardless of the setting of AB_LAYOUT_EXACT.
-hosts *computer_url_unc* [*computer_url_unc* ...]	Specifies that you want each directory of the layout to be the value of AB_WORK_DIR on the computers specified by the *computer_url_unc* arguments, in URL or UNC format. The AB_WORK_DIR directories must be writable.
-exact *exact_url_unc* [*exact_url_unc* ...]	Specifies that you want each directory of the layout to be exactly the directories you named with the *exact_url_unc* arguments, in URL or UNC format. The *exact_url_unc* arguments can be directories or files. If an *exact_url_unc* argument is a file, the Co>Operating System drops the name of the file and uses the resulting directory. The *exact_url_unc* must be writable.
-workdir *workdir_url_unc* [*workdir_url_unc* ...]	Specifies that you want the directories in the layout to be a subdirectories named .WORK in each *workdir_url_unc*, in URL or UNC format. The *workdir_url_unc* must be writable.

AB_AGENT_COUNT sets number of agents on SMPs

The Co>Operating system uses processes called "agents" to start various operations of an Ab Initio graph. These operations include file-system operations (for example, creating working directories) and process management (for example, starting and monitoring the component processes). The Co>Operating system assigns file-system operations and component processes to agents through multifiles and layouts.

By default, the Co>Operating system creates one agent per computer. On large SMP systems, such as those with more than 10 processors per computer, the agent can become overloaded, which can result in an excessive start-up time for graph execution.

The configuration variable AB_AGENT_COUNT enables you to increase the number of agents per computer. For example, if computers have 16 processors, you may want to set AB_AGENT_COUNT to **4**, so that four agents can operate on each computer. In general, having one agent for four to eight processors on a computer is optimal.

One agent per computer

If AB_AGENT_COUNT is **1**, the default, Co>Operating System assigns agents in the following way:

1. Expands the multifile to identify the computers where the partitions of a multifile are stored.

2. Starts one agent for every computer mentioned in the expansion.

For example, if the expansion of a multifile with four partitions identifies **computer1**, **computer1**, **computer2**, and **computer2** as the computers storing the partitions, then the Co>Operating System starts two agents, one on **computer1** and the other on **computer2**.

Multiple agents per computer

If AB_AGENT_COUNT is more than **1**, the Co>Operating system assigns file system operations in the following way:

1. Expands the multifile to identify the computers where the partitions of a multifile are stored.

2. Distributes the work for each computer among up-to-AB_AGENT_COUNT agents in round-robin fashion.

For example, suppose that AB_AGENT_COUNT is **2** and the expansion of a multifile with eight partitions identifies **computer1**, **computer1**, **computer1**, **computer1**, **computer2**, **computer2**,

computer2, and **computer2** as the computers storing the partitions. In this case the Co>Operating System starts four agents: two on **computer1** and two on **computer2**.

Agents are identified by the names of the computers on which they started, a percent (**%**) sign, and a three-digit number in sequence from **000** to **999**. For example:

The first agent on **computer1**:	**computer1%000**
The second agent on **computer1**:	**computer1%001**
The first agent on **computer2**:	**computer2%000**
The second agent on **computer2**:	**computer2%001**

These identifiers are called “dynamic aliases,” and may show up in various error messages, for example, if a component error is detected.

If AB_AGENT_COUNT is set to a number greater than the number of times a computer is mentioned in the expansion of a multifile, the number of agents is equal to the number of times the computer is mentioned. For example, if the expansion of a multifile contains the hosts **computer1** and **computer2**, then the system will create one agent for each host, regardless of the value of AB_AGENT_COUNT.

Avoiding bottlenecks with multiple agents

In some cases, the performance of the Ab Initio work directory can become a bottleneck. This is because all the agents started up on a given computer, by default, keep their rollback logs and other runtime structures in the Ab Initio work directory. The Ab Initio work directory is controlled by the configuration variable AB_WORK_DIR. Its default value is **/var/abinitio**.

To overcome this process, you can create multiple **abinitio** directories in different UNIX file systems and assign each agent to a different one.

For example, if **computer1** has file systems **/fs1** and **/fs2**, do the following:

1. Create the following directories:

   ```
   % mkdir /fs1/abinitio/vnode
   % mkdir /fs2/abinitio/vnode
   % mkdir /fs1/abinitio/cache
   % mkdir /fs2/abinitio/cache
   ```

   Users executing Ab Initio graphs must have read, write, and execute (search) permission.

You can set the right permission with the UNIX shell command **chmod**. For example:

```
% chmod 1777 /fs1/abinitio/vnode \
   /fs2/abinitio/vnode \
   /fs1/abinitio/cache \
   /fs2/abinitio/cache
```

2. Add the following lines to the Ab Initio configuration file:

```
AB_WORK_DIR @ computer1%000  : /fs1/abinitio
AB_WORK_DIR @ computer1%001  : /fs2/abinitio
```

Multiple agents with custom layouts

You may have assigned a custom layout to a program component. A custom layout specifies the directories you want the program component to use as working directories, instead of deriving them. To specify a custom layout for a program component in the GDE, you select **Custom** in the Layout tab of the Program Properties dialog box of a program component.

To create multiple agents per computer in custom layouts, you can use dynamic aliases in the URLs supplied in the custom layout.

Suppose, for example, that the partitions of a custom layout are specified as follows:

```
//computer1%000/part1
//computer1%001/part2
//computer2%000/part3
//computer2%001/part4
```

The Co>Operating System creates four agents: two on **computer1** and two on **computer2**.

AB_ALLOW_GRAPH_IN_COMPONENT

By default, if you attempt to run a graph (job) as or within a component of a graph, the graph fails and produces an error message like the following:

```
========= Error from ff.000 on hp4.abinitio.com =========
Running a graph as a component of another graph is not supported.
Aborting construction of job "m_ls" in component "ff.000".
You can turn off this check at your own risk by setting
  AB_ALLOW_GRAPH_IN_COMPONENT to true.
Aborting
```

An example of attempting to execute a graph within a graph is inserting an **m_** command in a component. The Co>Operating System considers the execution of the **m_** command a job in itself.

Running an Ab Initio graph within a graph is not supported. In some instances execution appears to work and in others it does not. If you do not want the graph to check for graphs within graphs, set the configuration variable AB_ALLOW_GRAPH_IN_COMPONENT to true. The default value is false.

AB_COMPATIBILITY and emulation of previous releases

The configuration variable AB_COMPATIBILITY enables the operation of Ab Initio software according to the behavior of a specific Co>Operating System release starting with release 2.1.11. The emulation does not extend to the internal protocols of the Co>Operating System, and so does not affect the restrictions described in "Limitation on compatibility" in the *Co>Operating System Release Notes*.

To operate according to the behavior of a specific release, set AB_COMPATIBILITY to the number of the release you want to emulate. For example, if you set AB_COMPATIBILITY to **2.1.22**, Ab Initio emulates Release 2.1.22 behavior.

AB_DATA_DIR specifies temporary file directory

The configuration variable AB_DATA_DIR specifies the directory for temporary data files that lack a specific, writable path in their layouts.

During job execution, components may need to write temporary data to disk. This can occur for large sorts, as a means of passing data across phase boundaries, for debugger files, and so on. In most cases, temporary files are maintained in the **.WORK** subdirectory of a component's layout. But when you use host layouts or database layouts, which both lack specific, writable paths in their layouts, or when you use unwritable layout directories, AB_DATA_DIR provides disk storage for the temporary files.

By default, AB_DATA_DIR is set to **/~ab_work_dir/data**.

- The initial /~ in the default value signifies a special Co>Operating System pathname.
- The value of **/~ab_work_dir** is the value of AB_WORK_DIR.

Starting with this release, the installation process automatically creates a **data** directory under **/~ab_work_dir**.

The default location is not ideal because the data files share the same file system as the recovery files and it is crucial that the recovery file system never fill up. If it were to fill up during a job, the job could be unrecoverable. To avoid this situation, use a different file system for AB_DATA_DIR. A RAID-5 file system is suitable for typical AB_DATA_DIR usage. If you need information about RAID-5, see your system administrator.

AB_DEFAULT_LOCAL_FLOW_TYPE

The flow type **usoc** (UNIX domain socket) is available for local flows (flows between components running on the same computer). The local flow type is set via the configuration variable AB_DEFAULT_LOCAL_FLOW_TYPE.

The complete set of possible values is:

npipe	Named pipe (the default)
pipe	Same as **npipe**
tcp	TCP socket
usoc	UNIX domain socket

The **usoc** flows have lower overhead than TCP connections and offer better buffering than named pipes. Unfortunately, when the operating system's pool of buffer space for UNIX domain sockets is exhausted, the failure is abrupt and causes graph execution to fail. For this reason, **usoc** is not the default.

Please note that UNIX domain sockets rendezvous on local file system paths, not on port numbers. In a directory listing, a **usoc** looks like a named pipe with no permissions (`p---------`). When you use a **usoc** in an Ab Initio graph, it appears in each agent's working directory with a flow component name, for example **f2.000**.

AB_IO_SIZE and other configuration variables set size of I/O operations

The following configuration variables are available for specifying the size of I/O operations:

- AB_IO_SIZE is a synonym for AB_TCP_IO_SIZE and affects only TCP connections. AB_TCP_IO_SIZE defaults to 16384 (16 Kbytes).
- AB_NPIPE_IO_SIZE is for named-pipe flows. Its default size is 8192 (Kbytes).
- AB_FILE_IO_SIZE is for file flows. Its default is 65536 (64 Kbytes).

Non-standard telnet ports

The configuration variable AB_TELNET_PORT specifies a non-standard port to use for telnet connections.

- If non-zero, AB_TELNET_PORT specifies the connection port for telnet.
- If AB_TELNET_PORT is zero, the Co>Operating System uses the standard or locally-configured telnet port.

Use the following form to set AB_TELNET_PORT:

```
"AB_TELNET_PORT@myhost.mydomain.com:1234"
```

See also "AB_TELNET_PORT and AB_FTP_PORT specify non-standard ports" on page 114.

Non-standard ftp ports

The configuration variable AB_FTP_PORT specifies a non-standard port to use for FTP.

- If greater than zero, AB_FTP_PORT specifies the connection port for ftp.
- If AB_FTP_PORT is zero, the Co>Operating System uses the standard or locally-configured ftp port.

Use the following form to set AB_FTP_PORT:

```
"AB_FTP_PORT@myhost.mydomain.com:1234"
```

See also "AB_TELNET_PORT and AB_FTP_PORT specify non-standard ports" on page 114.

AB_MAX_TRACE_FILE_SIZE sets size limit for trace files

You can set size limit in bytes for trace files produced by **XXtrace** by setting the environment variable AB_MAX_TRACE_FILE_SIZE. Any time the trace file exceeds the value of the variable, the software discards the first half of the file.

You must set AB_MAX_TRACE_FILE_SIZE in the environment, not in a configuration file.

The value of AB_MAX_TRACE_FILE_SIZE is an integer. The default value is **10000000**.

AB_REPORT

Specifying a percentage done column

To include a "percentage done" column after the "bytes done" column in a text-based tracking report, set **file-percentages** as one of the values of the configuration variable AB_REPORT. For example:

```
$ export AB_REPORT="monitor file-percentages"
$ mp run
--------------------------------------------------------------------------------
Dec 13 15:47:04   Phase 0 started (0 seconds)
      Data Bytes    (%)        Records      Status      Skew Flow Vertex   Port
--------------------------------------------------------------------------------
Dec 13 15:47:04   Phase 0 ended (0 seconds)
      Data Bytes    (%)        Records      Status      Skew Flow Vertex   Port
              36 (100%)             36 [    :    :   2]  38% f1   bc       in
--------------------------------------------------------------------------------
$
```

Numbers for bytes done and percentage done appear only for flows that read from files or multifiles.

See also "file-percentages" on page 59.

Job-monitoring reports

When the AB_REPORT configuration variable contains **summary=***filename*, a summary dataset is created at the end of a job, containing information about elapsed time, runtime, and data volume.

For further information, see "summary" on page 61 and "Summary reports" on page 73.

AB_STRUCTURED_ERRORS

The AB_STRUCTURED_ERRORS configuration variable controls the type of format in which the Co>Operating System emits errors.

- If AB_STRUCTURED_ERRORS is undefined, the Co>Operating System emits all error messages in text format.
- If AB_STRUCTURED_ERRORS=1, the Co>Operating System emits context-specific error messages in machine-readable format.

  To read error messages generated in machine-readable format invoke the command-line utility **m_view_errors**.

AB_TELNET_PORT and AB_FTP_PORT specify non-standard ports

You can specify non-standard telnet or ftp ports by setting the configuration variables AB_TELNET_PORT or AB_FTP_PORT in your own **.abinitiorc** file in your home directory or in the installation-wide one in **$AB_HOME/config/abinitiorc**.

For further information about **.abinitiorc** and the installation-wide configuration files, see "Notes on administration" on page 133.

Non-standard telnet ports

The configuration variable AB_TELNET_PORT specifies a non-standard port to use for telnet connections.

- If non-zero, AB_TELNET_PORT specifies the connection port for telnet.
- If AB_TELNET_PORT is zero, the Co>Operating System uses the standard or locally-configured telnet port.

Use the following form to set AB_TELNET_PORT:

```
"AB_TELNET_PORT@myhost.mydomain.com:1234"
```

Non-standard ftp ports

The configuration variable AB_FTP_PORT specifies a non-standard port to use for FTP.

- If greater than zero, AB_FTP_PORT specifies the connection port for ftp.
- If AB_FTP_PORT is zero, the Co>Operating System uses the standard or locally-configured ftp port.

Use the following form to set AB_FTP_PORT:

```
"AB_FTP_PORT@myhost.mydomain.com:1234"
```

AB_XFR_PROFILE_LEVEL and transformation engine profiling

Setting the configuration variable AB_XFR_PROFILE_LEVEL causes the system to profile the transformation engine at the specified level and gather information on performance of transformation code. When the variable is set to a supported value, a component that supports logging produces a record with a profiling report on its log port. The contents of the report depend on the value of AB_XFR_PROFILE_LEVEL.

The supported values of AB_XFR_PROFILE_LEVEL are:

function	Causes the transformation engine to record information for each transform function called. The profiling report lists each function, the number of times it was called, and the total time spent in the function and its callees.
statement	Causes the transformation engine to record the information recorded by **function** plus information for each local variable definition, statement, and rule. The profiling report lists the body of each function, line by line, with a count of the number of times each line executed successfully and the total time spent executing that statement.

Log records holding profile information have **event_type** "profile." The report appears in the **event_text** field, and is formatted in such a way that it is fairly easy to understand, but is also machine-readable, and can therefore be processed as data, for example, to create a profiling summary.

Caveats

Keep the following in mind:

- Enabling profiling can have a measurable effect on performance; statement level profiling may add as much as 50% to execution time.
- Line-by-line listing in reports is not exactly the same as source code; in particular, block structure appears with indentation only, and some constants are cast to a more efficient form.
- Statement level profiling for rules is not entirely accurate: profiling does not accumulate the count and time spent executing rules that result in NULL.

DML that describes the lines of the profile report is available in $AB_HOME/include/profile.dml:

```
type profile =
record
   string(' ') name;
   decimal(' times ') count;
   decimal(' secs ') time;
   string(' ') kind;
   decimal(':') index;
   string('\n') text;
end;
```

Examples

The following are examples of using AB_XFR_PROFILE_LEVEL.

Example 1

With AB_XFR_PROFILE_LEVEL set to **function**, a normalize component driven by the following package:

```
out :: length(in) =
begin
   out :: 2;
end;

out :: normalize (in, idx) =
begin
   out.key  :: in.key;
   out.dat  :: in.datavec[idx];
end;
```

Produces a profiling report (inside the **event_text** of a log record) that looks like the following:

```
length        4 times     0.020 secs func 0: length
normalize     8 times     0.010 secs func 0: normalize
```

Example 2

With AB_XFR_PROFILE_LEVEL set to **statement**, a reformat component driven by the following transform function:

```
out :: refm(in) =
begin
   let int i = 100;
   let int j = 100;
   while (i > 0)
      begin
         i = i - 1;
         if (i % 2)
         j = j - 1;
      end

   out.key :: in.key;
   out.a :: if (!is_blank(in.a)) in.a;
   out.b, out.c :1: if (!is_blank(in.b) &&
      !is_blank(in.c)) in.b, in.c;
   out.b, out.c :2: "bee", "cee";
   out.d :: 42;
end;
```

Produces the following profiling record on its log port:

```
refm          6 times    0.000 secs func 0: refm
refm          6 times    0.000 secs locl 1: let i = 100;
refm          6 times    0.000 secs locl 2: let j = 100;
refm        606 times    0.000 secs stat 3: while ((i>0))
refm        600 times    0.010 secs stat 4: i = (i-1);
refm        600 times    0.000 secs stat 5: if ((i%2))
refm        300 times    0.000 secs stat 6: j = (j-1);
refm          6 times    0.000 secs rule 7: out.key :: in.key;
refm          4 times    0.000 secs rule 8:
                             out.a :: (if (!is_blank(in.a))in.a);
refm          3 times    0.000 secs rule 9:
                             out.b, out.c :1: (if ((!is_blank(in.b) &&
                             !is_blank(in.c)))   in.b), in.c;
refm          1 times    0.000 secs rule 10: out.b, out.c :2: "bee", "cee";
refm          4 times    0.000 secs rule 11: out.d :: "42";
```

Example 3

The following package in a Normalize component reformats a log containing profiling information. Here, we compute an average time for each profiling event (by dividing the time by the count), and do some other simple reformatting.

```
// Include description of profile format:

include "~$AB_HOME/include/profile.dml";


// We'll hold the lines of the report
// in this "stringv", which is a
// vector of strings delimited
// by the string "END".

type stringv = string('\n')[delimiter == 'END'];

let stringv report_lines =
   make_constant_vector(0, '');


// Only look at records with event_type "profile."

out :: input_select(in) =
begin
   out :: in.event_type == 'profile';
end;


// In length, we cook up a stringv
// of the profile report, and then
// count the number of elements.
// That's how many output records
// we'll produce.

out :: length(in) =
begin
   report_lines =
      reinterpret_as(stringv,
      string_concat(in.event_text,
      "END\n"));
   out :: length_of(report_lines);
end;


// In the normalize function,
// we reinterpret each line
// as a profile record,
// compute an average time spent on each item,
// and do some other reformatting.

out :: normalize(in, index) =
begin
   let profile prof =
      reinterpret_as(profile,
      report_lines[index]);
   let decimal(' '.3) avg = 0;

   if (prof.count > 0)
      avg = prof.time / prof.count;

   out.name  :: string_lrtrim(prof.name);
   out.kind  :: string_lrtrim(prof.kind);
   out.index :: string_lrtrim(prof.index);
   out.count :: string_lrtrim(prof.count);
   out.time  :: string_lrtrim(prof.time);
   out.avg   :: avg;
   out.text  :: prof.text;
end;
```

Enhancement to reduce creation of adaptor processes

The Co>Operating System automatically inserts "adaptor processes" when it needs to move data from one computer to another and a direct TCP/IP connection is not possible. This happens when a non-local flow does one of the following:

- Crosses a phase boundary
- Accesses a data file
- Attaches to a custom component with non-soc ports

The Co>Operating System automatically checks the host-alias file when deciding whether an adaptor is needed.

On systems with multiple interfaces (for example, IBM SP systems), you need to identify a "primary hostname" for each "secondary interface." This is done using the configuration variable AB_HOSTNAME. In general, the primary host should be the one which is most widely accessible on the network, for example, an ethernet address in preference to a switch address. However, at this point no harm results from choosing the "wrong" interface.

For example, if you had two interfaces, **interface1** and **interface2** on the same host, you would designate one (for example, **interface1**), as being primary and put the following entry into the configuration file (for example **$AB_HOME/config/abinitiorc**):

```
AB_HOSTNAME     @ interface2     : interface1
```

If you had a third interface **interface3** you would also put the following:

```
AB_HOSTNAME     @ interface3     : interface1
```

Do *not* put extra "backward" entries in the configuration file. For example:

```
AB_HOSTNAME     @ interface2     : interface1 # Don't do this ... no
AB_HOSTNAME     @ interface1     : interface2 # backward entries
```

Do NOT construct "chains" of hostname declarations:

```
AB_HOSTNAME     @ interface2     : interface1 # Don't do this either ..
AB_HOSTNAME     @ interface3     : interface2 # No chaining
```

SSH connections

SSH connections are available on computers executing graphs on supported Hewlett-Packard, Sun Solaris, IBM AIX, Silicon Graphics IRIX, DIGITAL UNIX/Compaq Tru64, and Linux platforms. Please contact **support@abinitio.com** if you need SSH support for a platform not listed here.

To use SSH connections, set the AB_CONNECTION configuration variable to **ssh**.

The following other configuration variables are related to SSH.

- AB_SSH

  By default the **ssh** program is the one found in your path. If you want to specify an explicit path, set AB_SSH to the path.

- AB_SSH_NEEDS_PASSWORD

  If AB_SSH_NEEDS_PASSWORD is true, the SSH connection requires a password. If it is false, the connection does not require a password. The default is true.

- AB_PASSWORD

  Set AB_PASSWORD to the password you want the connection to use, if AB_SSH_NEEDS_PASSWORD is true.

- AB_CONNECTION_EXTRA_ARGS

  When using the **ssh** (as well as the **rsh**, **telnet** or **rlogin**) connection methods, you can pass special flags or arguments required by the connecting program by setting the configuration variable AB_CONNECTION_EXTRA_ARGS. This is not normally required. We recommended that you contact **support@abinitio.com** if you are having difficulties and suspect that you need to set this configuration variable.

DCOM connection method

The **Windows Native (DCOM)** connection method works only between Windows computers. In the GDE, to specify Windows Native as the connection method, you must have Release 1.11 or higher of the GDE. In the GDE, to specify DCOM as the connection method, you must have Release 1.11 or higher of the GDE.

Advantages

The following are the advantages of the Windows Native (DCOM) connection method over **rexec**:

- Specifying a username and password to connect is unnecessary.

  Windows security does the authentication and authorization on remote computers using the credentials of the currently logged in user. The Co>Operating System does not need to store or transmit passwords over the network. In fact, you cannot specify a username or password different than those of the current user.
- Configuring access to the application server is done with **dcomcnfg.exe**, the DCOM configuration utility provided with Windows.
- Any networked Windows computers can use DCOM.

  DCOM is available between Windows computers that are visible to each other through Windows networking. This resolves situations where, for example, DNS problems prevent the use of **rexec**.

Setting up DCOM

On the target Co>Operating System computer:

1. Check that DCOM is enabled.

DCOM is enabled by default. To make sure it is still enabled, run **dcomcnfg.exe**. The Distributed COM Configuration Properties dialog box opens. Click the Default Properties tab. Check **Enable Distributed COM on this computer** if it is not already.

2. Check that the list of users with DCOM permission is correct.

When first installed, the Co>Operating System DCOM application server allows only the users in the launch-permissions list to start it. Initially, this list contains only Administrator users.

To change this list of users, click the Applications tab of the Distributed COM Configuration Properties dialog box and then double-click **Ab Initio Exec Server**. The Ab Initio Exec Server Properties dialog box opens. Click the Securities tab. Select **Use**

custom launch permissions and then click **Edit** to display the list of users and edit the list if necessary.

A common strategy is to create a user group that you populate with users or domain groups to whom you want to give access, and then put this group into the custom launch-permissions list. For example, you could create a group called **Ab Initio Users**, add some users to it, and put this group in the list to give just those users permission to launch to the Co>Operating System DCOM server.

DCOM security

By default, Windows security prevents you from connecting to a target computer through DCOM and then accessing network resources from that target computer. For this reason, by default, you cannot connect the host to processing computers with DCOM if you have already used DCOM to connect the GDE computer to the host. Also, for this reason, you cannot access files on network servers from the target computer.

To get around this, do the following:

1. Run **dcomcnfg.exe** on the target computer.

   The Distributed COM Configuration Properties dialog box opens.

2. Click the Applications tab and double-click **Ab Initio Exec Server**.

   The Ab Initio Exec Server Properties dialog box opens.

3. Click the Identity tab and select **This User**.

4. Make entries in **User** and **Password**.

   The DCOM server then starts as this user, and can then "hop" to another machine. Note that the security check to see if the user is initially allowed to start the DCOM server is done against the connecting user, not against the user entered in this dialog. Also, the user entered in this dialog must have the "Log on as a batch job" security privilege. By default, Administrators have this privilege.

Specifying Windows Native (DCOM) connections

You can use Windows Native (DCOM) to connect the GDE computer connect to the host and the host to processing computers.

Connecting the GDE computer to the host

To connect the GDE computer to the host, do so in the GDE. Select **Windows Native (DCOM)** as the connection method on the Host Profile dialog box. To open the dialog box, select **Run**>**Settings**, click the Host tab, and then click **Edit**).

Selecting **Windows Native (DCOM)** disables the Login and Password fields on the Host Profile dialog box, because only the currently logged-on user can connect.

Windows Native (DCOM) requires that the computer being connected to is a Windows machine running Release 2.11 or higher of the Co>Operating System.

Connecting the host to processing computers

To connect the host to processing computers, do so in a Co>Operating System configuration file, for example, **abinitiorc**. Set AB_CONNECTION to **dcom**. You do not need to specify a username or password. In fact, you cannot enter a user different than the one you are logged in as. For example, a valid entry would be:

```
AB_NODES @ winhost : joe joe.acme.com 10.50.20.113
AB_HOME @ winhost : C:/AbInitio
AB_WORK_DIR @ winhost : C:/AbinitioWork
AB_CONNECTION @ winhost_all : dcom
```

A new utility **abntexec.exe**, which is similar to **rexec**, uses the DCOM connection method to connect to a remote computer and run a command. You can use this utility to test DCOM connection.

UNC filename syntax

On Windows, you can use UNC (Universal Naming Convention) syntax to refer to files on shared Windows volumes on servers.

The native UNC syntax is:

\\server\share\path

To specify a UNC to the Co>Operating System, use one of the following forms of URL:

unc://*fileserver*/*share*/*path*
unc://*abserver*//*fileserver*/*share*/*path*
—where *fileserver* and *abserver* must be computers running Windows. *abserver* (which defaults to the current computer) must have the Co>Operating System installed. *fileserver* will typically not have the Co>Operating System installed.

The syntax requires that you use the file type prefix **unc:**. If you do not use **unc**: the Co>Operating System will interpret the file reference as a standard URL of the form **//ab-initio-computer/path** and attempt to connect to **ab-initio-computer** using Co>Operating System URL protocols rather than Windows networking.

Use either backslashes or forward slashes to separate *server*, *share*, and *path* and within the *path* itself. For example, the following are equivalent:

```
unc:\\dept3fileserv\data2\billing.dat
unc://dept3fileserv/data2/billing.dat
```

Note that, although we support UNCs, we strongly discourage using them unless using them is unavoidable and then only for reading data files. DO NOT use UNCs to write to files on network drives. Reading data over the network can severely hinder performance and can be unreliable.

Configuration file names

The Co>Operating System accepts either of two names for the per-user Ab Initio configuration file. In addition to **.abinitiorc**, the Co>Operating System now also accepts **abinitio.abrc** in order to conform to Windows file name conventions. Other supported platforms also recognize the new name. Only one configuration file is permitted, however. Using both **.abinitiorc** and **abinitio.abrc** results in an error.

For more information on the abinitiorc file, see "Notes on administration" on page 133.

Searching for configuration files on Windows

On Windows platforms, the Co>Operating System searches for the per-user Ab Initio configuration file in $HOME as well as in $USERPROFILE/Personal. Previously on Windows, the Co>Operating System looked only in $USERPROFILE/Personal.

Note that only one configuration file is permitted. If the Co>Operating System finds more than one configuration file (**.abinitiorc**, **abinitio.abrc**, or both) in either or both directories, an error results.

On other operating systems, the Co>Operating System still searches only one directory: $HOME.

Debugging

The Ab Initio debugging system includes watchers, isolation mode, and context-specific error reporting.

- Watchers

  Watchers add intermediate files to flows automatically during graph development to view the data in the flow. The use of watchers mimics the well known debugging technique of manually creating intermediate files and wiring them into the graph. Watchers are available only in the GDE. For information, search for "watcher" in the GDE Help index.

  Watchers work with releases of the Co>Operating System earlier than Release 2.10, but the earlier releases do not support all layouts. An attempt to use a watcher in a place where it is not supported causes the Co>Operating System to emit an error message upon compiling and running the graph.

- Isolation mode

  Isolation mode identifies a subset of a graph and enables execution of only that subset. This makes debugging faster since it allows for rerunning a subset of a graph over and over without having to rerun the whole graph. Using Isolation Mode mimics the well known debugging technique of copying and pasting a subset of the graph to a separate graph during development. Isolation mode is available only in the GDE. For information about using isolation mode, search for "isolate selected components" in the GDE Help index.

  Isolation mode works with releases of the Co>Operating System earlier than 2.10, but the earlier releases support only file or multifile layouts. An attempt to use isolation mode with an unsupported layout causes the Co>Operating System to emit an error message upon compiling and running the graph.

- Context-specific error reporting

  Context-specific error reporting provides more details about errors, including solutions that are likely to resolve errors, than the error messages of previous releases. In the GDE, context-specific error messages are in color in the Job Output dialog. The GDE allows navigation from error message to component and back.

  Context-specific error reporting is available when using Release 2.10 or later of the Co>Operating System with Release 1.10 or later of the GDE.

For information about using the new error reporting system in the GDE, search for "context-specific error message" in the GDE Help index.

To display error messages in the SDE, use the new command-line utility **m_view_errors**:

The syntax of **m_view_errors** is as follows:

> **m_view_errors** [**-help**] [**-quiet**] [**-terse**] [**-verbose**]
> [**-errnum**] [**-noerrnum**] [**-read_until** *string*]

Following are the arguments to **m_view_errors**:

-help	Prints the usage statement for the utility.
-quiet	Suppresses warnings.
-terse	Shows only basic error messages, corresponding to the usual text-based error messages (DEFAULT).
-verbose	Shows full, verbose details on each error.
-errnum	Shows error numbers (DEFAULT).
-noerrnum	Hides error numbers.
-read_until *string*	Continues reading input until the utility encounters *string*.

Ticketing for performance tuning

Ticketing is a performance-tuning feature that reduces the contention for CPU cycles that can occur when a single Ab Initio job spawns many program components on a single computer. Reduced CPU contention can reduce swapping, thereby improving system performance, but it does not reduce total memory requirements.

For ticketing purposes, a running component is in one of two states:

- Computation state
- I/O state

Ticketing operates by restricting the number of components in a single job that can be running on one machine in the computation state at the same time. Before starting to execute in a computation state, a component must acquire one of a limited number of tickets. Tickets are released at the next flow I/O operation (reading from an input flow or writing to an output flow) or upon exit. Ticketing avoids introducing any new possibility of deadlock in Ab Initio jobs. Ticketing throttles only Ab Initio components, not third-party components.

Running in a non-computation state is "free," that is, it does not require a ticket.

Three configuration variables are associated with ticketing.

- AB_TICKETS specifies the number of tickets you want available per machine in the current job. The default value is **0**, which disables ticketing completely.

  Ticketing has very little effect on performance if you set the number of tickets to more than the number of processors. If you set the number of tickets equal to the number of processors, ticketing should reduce processor contention without slowing the job down. Values for AB_TICKETS that are less that the number of processors progressively throttle back computation.

- AB_COPY_USES_TICKETS specifies whether you want copy components, which do no computation, to be ticketed as they move from one I/O operation to the next. The default is false, which excuses them from ticketing.

- AB_SORT_YIELD_COUNT sets the interval at which a sort component must yield its ticket and acquire another one before proceeding. The value of the default interval is **100000** (one hundred thousand). The units of the interval are the number of key comparisons.

IBM OS/390 information

This section contains information about Ab Initio features that relate to IBM/390.

Specification of MVS datasets

The Co>Operating System handles MVS datasets as input, output, intermediate and lookup files. An MVS dataset is specified to the system using a URL of the form:

mvs://*hostname*/*datasetname,allocation_option*

hostname	The name of the computer on which the *datasetname* exists.
datasetname	A valid MVS dataset name
allocations_options	List of options used in allocating the dataset. See "Supported allocation options" on page 131.

An MVS dataset URL is not case sensitive.

Partitioned datasets and Generation Data Groups are supported using the usual syntax, so mvs:BIGPDS(MEM) refers to member MEM in the dataset BIGPDS.

The *allocation_options* portion of a URL allows one to specify characteristics like record format, dataset organization, and so on. The syntax is designed to be very similar to the TSO ALLOCATE command. So, to allocate a new output dataset named JOBOUT using 20 cylinders, with fixed block record format or 80 byte records, one would specify the following URL:

```
mvs:JOBOUT,CYLINDER,NEW,SPACE(20),RECFM(FB),LRECL(80)
```

Concatenating input datasets

Input datasets may be concatenated by specifying the DATASET option for each dataset. The following URLs both refer to the concatenation the two datasets DATA1 and DATA2.

```
mvs:DATA1,DATASET(DATA2)
mvs:,DA(DATA1),DA(DATA2)
```

Using generation data groups

There a few things to be aware of with the use of generation data groups in a Co>Operating System graph. The Co>Operating System permits a graph to read from an earlier generation and write to a future generation in the same phase of a graph. However, all generation numbers are relative to the phase of the graph. For example, suppose we have the following datasets in the GDG MYGDG:

```
MYGDG.G0003V00
MYGDG.G0004V00
MYGDG.G0005V00
```

Suppose we have a graph that in phase 0 reads MYGDG(0) and writes MYGDG(+1). If we refer to MYGDG(0) in a later phase, we refer to the dataset written in phase 0. So, in the above example, in phase 0 we read MYGDG.G0005V00 and write MYGDG.G0006V00. In subsequent phases, MYGDG(0) refers to MYGDG.G0006V00.

Generation data groups with generation number +1 are not usable as intermediate files, because the meaning of generation +1 changes between phases.

The MVS file components

We have added components specifically for handling MVS datasets as input, output, intermediate or lookup files. These components make it easier to use MVS datasets by providing a convenient place to put allocation options, as well as preventing the user from selecting meaningless options like file protection. The normal dataset Infile, Outfile, and do on can also be used if desired.

Because of the support for the allocation options below, we no longer require the use of special components to read and write tapes or create MVS datasets.

Support for VSAM datasets

For reading VSAM datasets sequentially, simply use an Input File or MVS Input File component. For updating, inserting, deleting or lookup up records in a VSAM dataset, use the VSAM_Lookup component.

Supported allocation options

For details on the meaning of these allocation options, see the help for the TSO ALLOCATE command.

Commonly used options are:

DATASET,DSET,DSN,DSNAME,DA The dataset name, for example DATASET(A.B.C)		
DDNAME,F,FILE,DDN,DD The DD name, for example FILE(FORT0001)		
DUMMY	BLKSIZE,BLOCK,BLK	LRECL
RECFM,RECF	DSORG	DSNTYPE
AVBLOCK	AVGREC	CYLINDER,CYL
SPACE	TRACK,TRK	DIR

ROUND	NEW,OLD,MOD,SHR	DISP
RELEASE,REL	UNIT	

Less commonly used options are:

DATACLAS	DENSITY,DEN	BUFNO
DEST	EXPDT	FILEDATA,FDAT
HOLD	KEYLEN	KEYOFF
LABEL,LAB	LIKE	LOCAL_GDG
MAXVOL	MGMTCLAS	NCP
NOHOLD	NO_MOUNT	OFFLINE_OK
POSITION,POS	RECALL	RECORG
RETPD	REUSE	RLS
STORCLAS	SYSOUT	TRTCH
UCOUNT	VOLUME,VOL	VSEQ
WAIT_FOR_UNIT		

A>

Notes on administration

This appendix contains detailed information for system administrators on these topics:

- Configuring the Ab Initio environment with configuration variables and environment variables.
- Configuring a heterogeneous computing environment.
- Configuring a system for high-availability clustering.
- Performance tuning an IBM SP-2 system.

Configuring the Ab Initio environment

This section lists Ab Initio environment variables that are set, typically at installation time, by the site system administrator. In this context, they are referred to as configuration variables.

Individual users may at times want to override some of configuration values in their own environments. Users may set the values locally as environment variables.

The variable settings

The system administrator may use the configuration file **$AB_HOME/config/abinitiorc** to set the variables listed below.Variables that are set in this file are called configuration variables and are visible systemwide. Any given shell may reset these variables as environment variables; in that case, the setting affects only the local shell.

Checking the values

The command **m_env** displays the current settings of the Ab Initio environment (or configuration) variables. Invoke **m_env** with the option **-h** for added help.

```
m_env -h
```

Note on system behavior

Environment variables are passed downward only. When a script is invoked, its environment variable settings will initially be those of the system configuration, but may be overridden by values in the shell that invoked it. If the script changes any of the values, those values are changed only for the course of the application. They are not changed in the invoking shell.

Ab Initio software location

As noted in Chapter 1, two environment variables must be set before any Ab Initio commands can be used: The system administrator cannot set these in **$AB_HOME/lib/abinitiorc**, since they are the means by which individual shells locate the Ab Initio software (including the configuration file). However, the administrator may wish to use some local mechanism to ensure that all users have these variables set to the proper value.

AB_HOME=*path*	
	The home directory of the installed Ab Initio software, usually **/usr/local/abinitio**. It is not necessary to set this variable on the remote hosts used by the application. The Co>Operating System will set it on all remote hosts to the same value it has on the host where the application script runs.
PATH=*path*	
	Search the **bin** directory under the home of the installed Ab Initio software; the value is usually **$AB_HOME/bin:$PATH**.

Temporary storage at run time

When executing applications, the Ab Initio system uses a temporary storage directory. Its default pathname is **/var/abinitio**. To specify a different directory, set the AB_WORK_DIR configuration variable to the desired pathname.

The temporary storage directory must exist before an Ab Initio program can run, and the directory must contain subdirectories named **host**, **vnode**, and **cache**. By default, the installation script creates these subdirectories under **/var/abinitio**; if you relocate the root of the temporary storage directory, be sure to create the three subdirectories in the new location.

Systems with multiple networks

The following environment variable applies to systems with multiple networks.

AB_HOST_INTERFACE=*hostname*	
	Name of the interface to use for host-host communications on systems that have multiple networks (and thus multiple hostnames for each host). See page 15 for explanation.

Remote connections and heterogeneous environments

The following environment variables apply to remote connection and heterogeneous environments.

AB_RSH	
	Indicates the full pathname of the location of the **rsh** command. It defaults to the standard location for **rsh**, for example, **/usr/bin/rsh** on most UNIX platforms, **/bin/remsh** on HP-UX, **/usr/bin/resh** on Sequent.

AB_CONNECTION, AB_CONNECTION_SCRIPT, AB_PASSWORD, AB_USERNAME	
	Control aspects of remote connections. See discussion below, on page 138.
AB_EXPORT, AB_CONFIGURATION	
	Used to centralize the settings of configuration variables. See discussion on page 146.

System behavior

Set these environment variables if you want a value different from the default:

AB_TIMEOUT=seconds	
	The time-out interval for certain operations, such as starting up a remote process. Default is **30** seconds.
AB_MAX_RECORD_BUFFER=bytes	
	Maximum buffer size that certain parts of the system will use to hold a record. Default is **5000000** (5 million) bytes. Note: Incorrect metadata or corrupt data may cause data to appear to the system as an unreasonably large record. When the space needed to buffer a record exceeds the value of AB_MAX_RECORD_BUFFER, the system reports an error and aborts execution.

AB_NICE=priority	
	Run jobs on remote hosts at the specified priority. The values reflect UNIX priorities, which range from **-20** (highest) to **19** (lowest). A user’s shell typically runs at priority **0**, and only the superuser can increase priorities (i.e., move to lower numerical values). Note: This value is usually set by the user on a per-job basis. As a result, the user can set AB_NICE to values from **1** (just a little nice) through **19** (exceedingly deferential). If an unattainable value is specified, the processes will run at the default priority. Note to Korn shell users: The shell transparently reduces the priority of any job that runs in the background (and the priority of any processes directly or indirectly created by such a job). This can degrade performance. To avoid this behavior, set. the **ksh** option **bgnice** to **+0**.
AB_DEFAULT_LOCAL_FLOW_TYPE	
	When a flow does not cross host boundaries (e.g., straight flows on an MPP; almost all flows on an SMP), then the Ab Initio software may use either TCP/IP or named pipes for data transport. This selection is controlled by the configuration variable AB_DEFAULT_LOCAL_FLOW_TYPE, which may have the values **tcp** or **npipe**.
	As of release 2.0.10, this configuration variable is a *heterogeneous* configuration variable rather than a *global* configuration variable. This means that it may take on different values for different hosts. For example, to use named pipes on the host called **smp-3** and the default TCP/IP on all other hosts, put this line in the Ab Initio configuration file: `AB_DEFAULT_LOCAL_FLOW_TYPE@smp-3: npipe`

AB_AGENT_ERROR_DELAY	
	The Ab Initio system terminates a job when one of its processes exits abnormally. It delays termination for **0.25** seconds to permit recovery of logging information, etc., from other processes in the job. The length of the "grace period". is controlled by the value of AB_AGENT_ERROR_DELAY (in seconds). This is a heterogeneous configuration variable, so it may be overridden on a per-host basis.

Special Ab Initio facilities

These facilities are described in more detail elsewhere in this manual.

AB_HOST_ALIAS_FILE=*path*	
	File containing hostname aliases. See page 29. Note: This variable may be set by the user on a per-job basis
AB_CATALOG=*path*	
	Location of user-created metadata catalogs. See page 32. Note: This variable may be set by the user on a per-job basis
AB_REPORT=*keyword*	
	Monitor the current job and produce reports. See "AB_REPORT settings" on page 58 for keyword values. Note: This variable may be set by the user on a per-job basis

Debugging

The following variables are usually set by the user as environment variables.

DISPLAY=*display-id*	
	The X Windows display. Used to pop up debuggers.

Logging/recovery control

The following variables may be set at the request of Ab Initio support staff to assist in tracking down a problem:

AB_NO_AUTO_ROLLBACK=	
	Suppress auto rollback of aborted jobs. See page 50 for explanation.

AB_RETAIN_LOGS=*keyword*	
	Delete or retain log files after successful completion of the application. By default, log files are deleted. The *keyword* is one of: • **yes** - retain log files • **“”** [no value] - retain log files • **auto** - delete log files

Configuring a heterogeneous environment

This section describes Ab Initio facilities that are used to control operation in a computing environment where all machines are not necessarily identically configured.

Locating custom components

The location of a custom component is indicated by a URL path for the image parameter in the component's **.mpc** file. Some special notations can be used to indicate the location in a portable fashion:

~**ab_home**	
	Same as **$AB_HOME** (e.g. **/usr/local/abinitio**)
~**ab_work_dir**	
	Same as **$AB_WORK_DIR** (e.g. **/var/abinitio**)
~$*config_var*	
	Expands to the value of the configuration variable *config_var*

Using the third form (and properly initializing *config_var*) makes it possible to place an executable anywhere. The location may be different on different machines.

Example

Here is an example of how the ~ notation in URLs can be used with configuration variables to tell Ab Initio software where to find the executable program for a custom component.

Assume that you have a custom component whose **.mpc** file is named **custom.mpc**, which contains:

```
image: "/~$CUST_LOCATION/mycomp"
exit: 0
port: soc out out
argument: literal $*
```

On machine **firstmach**, **mycomp** is in **/usr/local/ab/mycomp**, and on **secondmach** in **/usr4/blarney/bin/mycomp**.

On the machine where you’ll run mp, the **~/.abinitiorc** file will contain the following entries:

```
CUST_LOCATION @ firstmach : /usr/local/ab
CUST_LOCATION @ secondmach: /usr4/blarney/bin
```

NOTE: There are other variations possible in how to set configuration variables and in the files in which they are set. These are discussed later in this section.

If you run this command:

```
m_env -node-info firstmach
```

To see the settings that will be passed over to firstmach, the report will include the following:

```
linux8# m_env -node-info firstmach
...
Configuration Variable Settings for node foo:
    (use "m_env -describe" for descriptions)

Variable       Set    Value or Resulting Action
-----------    ---  -------------------------
AB_HOME         *     set to "/l8b/myname/src/2-0-7"
AB_WORK_DIR           default is /var/abinitio
AB_CONNECTION         default is rsh to remote
                         hosts;
                      exec to localhost
AB_USERNAME           default is current user (name)
CUST_LOCATION         /usr/local/ab
...
```

Similarly, the output of

```
m_env -node-info secondmach
```

Will include the line:

```
CUST_LOCATION         /usr4/blarney/bin
```

Configuration variables and heterogeneous environments

The behavior of Ab Initio software is controlled in part by a family of values known as configuration variables. Configuration variables are used to control debugging behavior, resource allocation, temporary file location, remote connection methods, and more. For example, the configuration variable AB_HOME identifies where Ab Initio executables and libraries are installed, and AB_WORK_DIR identifies where temporary job-related files will be stored.

In general, you can think of configuration variables as establishing a "context" in which Ab Initio software runs. Since Ab Initio jobs can run in parallel on different computers which may be differently configured and which may be running different operating systems, the contexts needed for the various computers participating in a job may not be identical. Configuration variables allow you to describe the contexts for different computers. Configuration variable settings can be personalized for individual users, established centrally for all users, or a combination of both.

Initialization

The first step taken when any Ab Initio software starts to run is to set its configuration environment by reading any initialization files. For each configuration variable named in an initialization file, Ab Initio first checks to see if that configuration variable is defined in the process environment. If it is, then the configuration variable is saved with the value from the environment. Otherwise, it is saved with the value in the initialization file.

Initialization files don't have to exist. When Ab Initio software uses a configuration variable that wasn't named in an initialization file, it checks the process environment and uses that value (if there is one), or falls back to a default. The shell command **m_env** provides information about some of the important configuration variables and their default values.

Duplicate entries for the same configuration variable are allowed. The value is set when the first entry is encountered, and all subsequent entries are ignored.

Initialization file names

The standard configuration files are:

```
~/.abinitiorc
$AB_HOME/config/abinitiorc
```

The environment variable AB_CONFIGURATION can be set to a colon-separated list of additional files. They will be read in the order given, after which the two files listed above will be read. For example, the setting

```
export AB_CONFIGURATION=file1:file2
```

Will result in reading

```
file1, file2, ~/.abinitiorc and $AB_HOME/config/abinitiorc
```

It does not matter if some or all of the files do not exist.

Configuration file syntax

Entries in initialization files can have the following forms:

- Comment lines are those where the first non-space character is #.
- Configuration variable lines have one of these forms:

  config_var : *value*

  or

  config_var **@** *host-or-alias* : *value*

  Where *config_var* is the name of the configuration variable and *value* is the value that it will take if *config_var* isn't defined in the environment. Normally the entire entry should be on a single line, but continuation lines can be used if the first non-space character on the continuation line is +. The second form (with **@** *host-or-alias*) is discussed below in the section on indirect configuration variables.
- Additional initialization files can be included with a line of the form:

  @include *filename*

  If *filename* cannot be found, a fatal error is reported.

Configuration variables for remote execution

Every time Ab Initio software starts, it checks for initialization. This is also true for software components that are executed remotely by **mp**. So if initialization files are being used to set configuration variables needed by components, it is important to be sure that the desired values are set in the initialization files located on the machine where the component is running.

But having to manage different initialization files on (possibly) many different computers can be both tedious and error-prone. An alternative is to use the centralized configuration facilities.

Remote connection

Certain configuration variables control how remote connections are established. These are:

AB_CONNECTION	
	One of **exec**, **rsh**, **rexec**, **telnet**, **rlogin**.
AB_CONNECTION_SCRIPT	
	Used by **telnet** and **rlogin**. Defaults are: • **$AB_HOME/lib/telnet.script** and • **$AB_HOME/lib/rlogin.script**
AB_HOME	
	Absolute directory path where Ab Initio is installed on the remote machine in the proper syntax for that machine.
AB_PASSWORD	
	Used for **rexec**, **telnet**, and **rlogin**. See page 146 for a more secure alternative.
AB_USERNAME	
	Account name to use for the connection.

These variables are necessary to establish remote connections, so they are needed by **mp**. You can use the indirect configuration variable notation (as discussed below) for these if the connection information is not the same for all machines.

Indirect configuration variables

Some configuration variables are used not by **mp** itself but rather to configure other processes that it invokes.

There are many cases when mp might want its remote processes to use different configuration values than it uses itself. A common example is AB_HOME, where some of the remote computers have Ab Initio software installed in a different location from the current machine.

Indirect configuration variables identify the targeted remote host (or hosts):

config_var **@** *host-or-alias* **:** *value*

For example, if hostname **bob** has Ab Initio installed in **/user5/local/ab** and **mp** is initialized with these configuration variable settings:

```
AB_HOME : /usr/local/abinitio
AB_HOME @ bob : /user5/local/ab
```

Then remote jobs on host bob will use **/user5/local/ab** for AB_HOME while all other hosts will use **/usr/local/abinitio**.

Aliases for hosts and groups of hosts

Sometimes, when several hosts are similarly configured, it is convenient to use an alias that identifies all the similar machines. A host alias can be defined with an AB_NODES configuration variable definition like this:

```
AB_NODES @ onehosts :  bob1 fred1 geo1
```

This example establishes the name **onehosts** as an alias for any of the hosts **bob1**, **fred1**, or **geo1**. Wildcards can be used to describe the matching hostnames:

```
AB_NODES @ nalias :  bob*  fred1 geo?
```

Pattern matching rules in aliases

The syntax of the patterns used in AB_NODES pattern lists follows these wildcard rules:

- ***** matches zero or more characters
- **?** matches any single character
- **[...]** matches any character inside the [ ]
- \ escapes any character in the pattern

Notes on indirect variables

If AB_USERNAME is different from the account name running **mp**, then the behavior of the connection methods that do not normally specify a username will be altered:

- **rsh**: The **-l** *user* option will be used for **rsh** connections.
- **exec**: Instead of using **exec** for local jobs, **mp** will use a remote connection method with the new account name.

It is not usually necessary to fully qualify hostnames with their domain name in the configuration variable file. But you should keep some things in mind:

- The wildcards used in AB_NODES never match "**.**".
- When **mp** searches the indirect configuration variables for settings for a particular host and the hostname is not qualified with the domain name (e.g., it appears as **bob1** rather than **bob1.mydomain.com**) and the search fails, it will append the local domain name and retry the search.
- You should be consistent in uses of the name for *host-or-alias*: use the domain name always or never. There is a chance of confusion and unexpected results if hostnames and aliases are mixed up.

Summary of configuration variables for remote connections

These configuration variables pertain to establishing remote connections. In all cases, the value *host-or-alias* can be either a hostname or an alias for several hosts.

AB_NODES **@** *alias* **:** *pat* [*pat*] ...	
	alias - A name that will be associated with hostnames that match the supplied patterns. *pat* - A wildcard expression (or just a simple hostname). See discussion of pattern matching, above.
AB_CONNECTION [**@** *host-or-alias*] **:** \\ (**rsh** | **rexec** | **telnet** | **login** | **exec**)	
	Use the connection mode named (it must be one of the choices shown) when making remote connections. Setting this configuration variable to, e.g., **rexec** will not prevent **mp** from using the (much) more efficient **exec** method when starting a process on the local machine.
AB_PASSWORD [**@** *host-or-alias*] **:** *password*	
	Establish the password used when connecting to the named host or aliased hosts (but only if the connection method requires a password). This is not an especially secure way to keep a password. An alternative is to use the **~/.netrc** file or not supply one in any file. If **mp** needs to have a password when connecting and does not have one, it will interactively prompt.

AB_CONNECTION_SCRIPT [**@** *host-or-alias*] **:** *path*	
	The **telnet** and **rsh** connection methods require a script that describes the sequence of steps needed to log in to the remote machine. By default **telnet** uses **$AB_HOME/lib/telnet.script** and **rlogin** uses **$AB_HOME/lib/rlogin.script**. This configuration variable allows you to use something else.
AB_USERNAME [**@** *host-or-alias*] : *user-name*	
	Names the user account to establish a connection with. If *host-or-alias* applies to the machine on which **mp** is running and *user-name* is different from the user account running **mp**, then it will cause **mp** to use a connection method other than **exec** (typically **rsh** unless something else was specified).
AB_HOME [**@** *host-or-alias*] : *path*	
	The location of the Ab Initio software installation on the named host. This must be expressed in the path syntax of the named machines.
variable [**@** *host-or-alias*] : *value*	
	Any configuration variable can be given an indirect definition. When establishing a remote connection, all the indirect configuration variables applicable to that host will be passed.
AB_EXPORT *var1***:***var2* ...	
	A colon-separated list of configuration variables whose value will be passed to the remote machine. These will be passed whether they have an indirect definition or not.
AB_CONFIGURATION *file1***:***file2* ...	
	A colon-separated list of files that define configuration variables.

Configuring a cluster for high availability

With high-availability clustered computing, the workload from a failed host can be shifted to other hosts in the cluster. This entry explains how to implement high-availability clustering with Ab Initio Software.

This entry presents the general principles of configuring a system for high-availability clustered computing and then addresses the specific issues involved. These issues are:

- Remapping file systems
- Remapping hosts
- Resource issues
- Restoring normal service
- Recovery issues

General principles

Ab Initio incorporates checkpoint/restart capabilities which allow it to automatically recover the state of a job after a system failure. In some cases, recovery will consist of cleaning up stray files and restarting the job from the beginning. In other cases, recovery will consist of rolling back the state of the system to a checkpoint then continuing the job from that checkpoint.

For these capabilities to be used, it is necessary to restore the system environment used by the job. In some cases this may be done by rebooting a host that has crashed. In other cases, this may be done by substituting different hardware for the failed host. This entry is concerned with the latter case: the high-availability case.

To implement a high-availability solution, we need to:

- Remap hosts. Substitutes must be provided for the failed hosts.
- Restore access to files. All files that were used by the application must be accessible from the substitute hosts.
- Restart the failed applications. The restart will automatically trigger Ab Initio Software's recovery mechanisms.

At a later time, the original hosts may be mapped back in.

Remap hosts and files

Ordinarily, Ab Initio identifies hosts using IP hostnames. The network protocols then map each hostname to a specific interface, which might be an Ethernet card, a high-speed switch adapter, a FIDDI card, etc. Host substitution may be done within the operating system by manipulating the routing tables that perform this hostname-to-interface mapping. IBM's HACMP is an example of this approach.

For cases where O/S host substitution is unavailable or not sufficiently flexible, Ab Initio includes its own host substitution mechanism. This is done by creating and manipulating a host alias file, as described below.

There are two areas of concern with regard to file access: data files and recovery files. Data files are generally managed by the Ab Initio multifile system. To implement a high-availability solution, the multifile system must be configured with fail-over modes in mind. In addition, it may be necessary to alter file system mount points during the host-substitution process. Recovery files are managed by Ab Initio, and are generally stored inside **/var/abinitio**. Again, high availability requires proper configuration of the file system and possible alteration of file system mount points. In addition, it will sometimes be necessary to alter symbolic links or relocate recovery files, as will be described below.

Automatic recovery

The Ab Initio system supports checkpoint/restart. In most cases, simply rerunning the application script will trigger the recovery mechanism, which will roll back all changes made by the script and then restart it from the beginning. This is completely automatic. If intermediate checkpoints are desired, then the application developer needs to insert **mp checkpoint** commands into the script (consult the *Ab Initio Shell Development Environment User's Guide* for further details). Again, all that is necessary is to restart the script; Ab Initio will automatically detect the existence of a checkpoint and pick up the job from there.

Non-automatic recovery

The only case where the application developer needs to be concerned with checkpoint/restart issues are:

- When the job alters a database, for example by loading data into a table.
- When the script contains multifile commands, for example, **m_mkdir**.
- When the script contains more than one **mp run** command.

  When the script contains UNIX commands that alter the file system, for example, **mkdir**.

Eventually, the failed host will be repaired or rebooted, at which point the work done in the host-substitution process needs to be undone.

For this discussion, we will assume that every file system has a unique name that is used as its mount point. For example, we might have two file systems **fsA** and **fsB** on storage devices shared by two hosts **hostA** and **hostB**. We will assume that **fsA** is mounted as **/fsA**, regardless of whether it is mounted by **hostA** or by **hostB**. Similarly, **fsB** will be mounted as **/fsB** regardless of which host it is mounted on. Other approaches are possible but they will not be discussed in this document.

All files and directories that might be modified by Ab Initio must be resident on shared mass storage. This includes all partitions (control and data) of all multifile systems plus the **/var/abinitio** directory where Ab Initio keeps recovery-related information. In addition, the application itself needs to be run from a directory on shared mass storage. Files and directories that are not modified by Ab Initio (for example, its own root directory) do not have to be on shared mass storage, provided they are replicated on each host.

Remapping file systems

The most important issue is proper configuration of multifile systems. When a multifile system is created, a hostname/directory pair is specified for the control partition and for each data partition. For example, suppose we have hosts **A**, **B**, and **C**, and suppose we have file systems **A**, **B** and **C** that will be used as control and data partitions. We could create the file system with the **m_mkfs** command:

```
m_mkfs //hostA/fsA/mfs //hostB/fsB/mfs //hostC/fsC/mfs
```

If we later have a failure and substitute a spare host **hostD** for one of the original hosts **hostC**, we must make sure that **/fsC** is mounted on **hostD**.

Rule for remapping data files

If a URL //**A**/*filename* is supplied to Ab Initio, and a second host **B** is substituted for **A**, then file-system mount points must be adjusted such that /*filename* will be valid on **B** and refer to the same files and directories as it formerly did on **A**.

This may generally be accomplished by simply adjusting mount points at host-substitution time.

A second issue is the configuration of the Ab Initio recovery directory **/var/abinitio**. This directory contains logs and recovery files required by Ab Initio's checkpoint/restart mechanism. If the recovery directory is not properly reconfigured at host-substitution time, then jobs that were active at the time of the failure will not be recoverable until the original host is restored. This will not, however, prevent the introduction of new jobs into the system.

Rule for remapping recovery directories

If a directory **/var/abinitio/vnode/***dir-name* was present on host **A** at the time of a failure, and **A** is remapped to **B**, then the file system must be adjusted such that **/var/abinitio/vnode/***dir-name* will be valid on **B** and refer to the same directory as it formerly did on **A**. The same rule applies to /**var/abinitio/host** and /**var/abinitio/cache**.

This is best accomplished by the following procedure:

- For each host, create a distinct recovery directory on shared disk.
- On each host, create a symbolic link from **/var/abinitio** to that host's recovery directory.
- At recovery time, either adjust the symbolic link or relocate the contents of the recovery directory.

Suppose we have two hosts **A** and **B** and two shared file systems **fsA** and **fsB**. We could set up the recovery directory for **A** by executing the following commands (these commands require **root** privileges):

Example of creating recovery directories

```
# Create the directories

mkdir /fsA/abinitio
mkdir /fsA/abinitio/vnode
mkdir /fsA/abinitio/host
mkdir /fsA/abinitio/cache

# Only root can alter abinitio.

chmod 0755 /fsA/abinitio
# Anyone can write to vnode, host, or cache,
# but the 'sticky' bit is set to prevent
# users from interfering with each other.

chmod 1777 /fsA/abinitio/*

# Create a symbolic link

ln -sf /fsA/abinitio /var/abinitio
```

A similar set of commands would be issued from host **B**.

At host-substitution time, there are two scenarios:

- The failed host is remapped to a hot standby.
- The failed host is remapped to an existing host.

The hot spare case

Suppose that host **A** fails, and we have a hot standby **C**. To recover, we would need to:

- Remap **A** to **C**
- Mount **/fsA** on **C**
- Adjust the symbolic link:

```
ln -sf /fsA/abinitio /var/abinitio
```

Note that these commands will generally require **root** privileges.

Remapping hosts

To implement a fail-over strategy we need some mechanism for mapping names to network addresses and for modifying that mapping after a failure.

In some cases this remapping may be done by the operating system. For example, IBM’s HACMP facility allows a logical service name to be mapped to a physical hostname. If the operating system provides a service-name mapping facility, then all hostnames given to Ab Initio must incorporate that service name. In particular, service names must be incorporated into all URLs given to the system, for example, in **mp** file commands and in shell-level multifile commands.

If a host has several network interfaces (for example, an Ethernet and a FIDDI), then only the network interfaces used by Ab Initio need to be supplied with service names and/or remapped.

Hostnames are likely to show up in the following contexts within Ab Initio:

- In URLs for multifile commands such as **m_mkfs** and **m_mkdir**.
- In URLs supplied to the following mp subcommands: **ifile**, **ofile**, **iofile**, **layout**.
- In hostnames supplied to the **mp layout** command
- In the metadata catalog commands
- In various configuration files created by the database access package.
- As the value of the AB_HOST_INTERFACE environment variable.

As a supplement/alternative to the operating system service mapping, Ab Initio makes provision for a host alias file that can be used for much the same purpose.

Format of host alias file

A host alias file contains lines of the form:

```
hostname1 alias1
hostname2 alias2
....
```

Set the environment variable AB_HOST_ALIAS_FILE to refer to that file. Every user of the Ab Initio system must set this environment variable. For example, in Korn shell you might type:

```
export AB_HOST_ALIAS_FILE=/admin/host-alias
```

This may be done when the job is started or when the job is restarted after a failure. This facility may be helpful in cases where the name of a failed host has somehow crept into the system.

The host alias file could, if desired, be used in combination with a service-name mapping facility.

Resource issues

In cases where a hot spare is not available, it is necessary to move the workload of the failed host onto existing hosts. This can lead to two problems:

- Slowdown

  If host **A** fails and we map **A**'s work to **B**, then **B** will have twice the workload that it formerly had, and the application will run at half its normal speed. This is true regardless of how many hosts are participating in a parallel job. For example, if a job uses 10 hosts, one of which is remapped, then the entire job will run at half speed even though only one host is double-loaded. This is because a parallel job only runs as fast as the slowest host.

- Main memory contention

  If we map **A**'s work to **B**, then **B** must have enough main memory to support **A**'s workload as well as its own. For example, if the workload on **A** requires 100 megabytes of main memory and the workload on **B** requires 100 megabytes of main memory, then **B** must have 200 megabytes of main memory in order to support the fail-over workload. If **B** does not have enough memory to support the increased demands, the system might begin to thrash.[1] This may be diagnosed by looking at the paging statistics produced by the vmstat utility. If you see hundreds of page-outs per second coupled with extremely poor performance, the system may be thrashing. This situation might cause the backup host to crash or to perform so poorly as to be useless.

  Under certain circumstances, it is possible to spread the fail-over load across more than one host. Specifically, if you have a cluster of three or more SMP hosts, then the load may be rebalanced after a failure.

In this context, a cluster is a group of hosts that *all* have access to a pool of shared storage devices. This access may be via a physical connection (for example, multihosted SCSI), or via a network (for example, NFS), or via some combination of the two. The key requirement is that all hosts in the cluster be able to access the files on those devices, in accordance to the rules stated in the section on file system remapping.

Suppose, for example, that we have three 2-way SMP hosts named **A**, **B**, **C**, plus six file systems **fsA1**, **fsA2**, **fsB1**, **fsB2**, **fsC1**, and **fsC2**. Further, suppose that each of these file systems can be mounted on any of the hosts.

Ordinarily, if **A** fails we will have to map it to one of the remaining hosts (**B** or **C**); that host will then have 200% of its normal load.

1. Thrashing is a situation where main memory is so badly overcommitted that the system spends essentially all of its time paging.

Suppose, however, that when we initially configure the system, we configure it as six virtual hosts: **A1**, **A2**, **B1**, **B2**, **C1**, and **C2**. This may be done by creating six service names in the O/S layer, or by creating six entries in the host alias file. We could then create a multifile system using the following command:

```
m_mkfs mfs   //A1/fsA1/mfs //A2/fsA2/mfs   \
             //B1/fsB1/mfs //B2/fsB2/mfs   \
             //C1/fsC1/mfs //C2/fsC2/mfs
```

This set of six service names gives us an easy way of specifying computations that are six ways parallel.

If **A** fails, we must remap the service names **A1** and **A2**. We can, however, remap them independently: we could remap **A1** to **B**, and remap **A2** to **C**. This results in **B** and **C** each having 150% of their normal workload, rather than 200%. This scheme may be extended to larger configurations, subject to the ability of the platform to support large clusters.

In this case, the procedures shown above for relocating the Ab Initio recovery directories require alteration. Specifically, when we split **A**'s workload across **B** and **C**, we have no way of telling which elements of **A**'s recovery directory need to be relocated to **B**, and which need to be relocated to **C**. The best solution to this problem is to copy **A**'s recovery directories to both **B** and **C**. After the job has been restarted and run to completion, it will be necessary to remove stray entries from the recovery directory.

Restoring normal service

When the failed host is back in service, it will be necessary to restore the original system configuration. This is primarily a matter of restoring the normal file system mount points and service name mappings. If a host alias file is employed, it must be restored to its normal configuration. The Ab Initio recovery directory must also be returned to normal service.

Ideally, restoration of normal service may be done when the system is idle and there are no unrecovered jobs. Under these circumstances, the contents of **/var/abinitio/vnode**, **/var/abinitio/host**, and **/var/abinitio/cache** may simply be removed. If symbolic links from **/var/abinitio** to local recovery directories have been altered, those links should be restored to their normal configuration.

If restoration of normal service is done when unrecovered jobs are outstanding, then the best course is to copy the contents of the replacement host's recovery directory into the original host's recovery directory, but not to delete anything in the replacement host's recovery directory. This will result in stray files being left in the replacement host's recovery directory; such stray files should be cleaned up periodically. Remember, it is always safe to clean out the recovery directories when the system is idle and there are no unrecovered jobs outstanding.

Recovery issues

For most jobs consisting of simple **mp** scripts, recovery is automatic and is handled entirely by Ab Initio. The first time the script is run, a small file with the suffix **.rec** is created. The **rec** file contains the name of a file within the Ab Initio recovery directory. When the job terminates normally, the **rec** file is removed. If a job terminates abnormally, it will first attempt to do a rollback. If the job is rolled back to its beginning, then the **rec** file will be removed. If, however, the job cannot be rolled back (for example, because a host has crashed), or if the job rolls back to a checkpoint, then the **rec** file will remain and the job is considered to be in an unrecovered state. When the job is restarted, the **rec** file will be detected and information stored in the recovery directory will be consulted to recover the job and, if necessary, restart the job from the most recent checkpoint.

In order for this recovery mechanism to work, the job should be restarted from its original directory, running under the user ID of whoever originally submitted the job. If a batch queuing system is in use, it is sufficient to configure the queuing system to resubmit any jobs that had not completed at the time of the host failure.

Special considerations for database applications

If the job modifies a database, it is necessary to carefully consider the recovery strategy. The primary issue is that Ab Initio jobs generally process such large amounts of data that it is not possible to do all the work as a single transaction. It is therefore not possible to truly roll back the job.

There are two strategies for coping with this problem:

- Put recovery handling into the mp script. For example, if a job does a bulk load into a table, and the table is known to be empty prior to the job, then it is reasonable to truncate the table before recovering a job. For example:

  ```
  if [ -f myjob.rec ] then truncate_table ; fi
  mp job myjob
  ....
  mp run
  ```

- Utilize components that are restartable. A restartable component will keep track of how much work has been committed to the database and skip over work that has already been committed. The Ab Initio database components are written this way. For example, the database loader might be configured to do a commit every 10,000 rows. If a system failure occurred after 133,753 rows had been processed, and we restarted the loader, it would know that the first 130,000 rows had been committed, and thus skip the first 130,000 rows of its input.

  If restartable components are used, then the order of their input data must be deterministic. In most cases, this can be accomplished by putting a checkpoint immediately upstream from the database component.

Recovery when multifile commands are used

If the script contains multifile commands such as **m_mkdir** or **m_mv**, then the user needs to use the **m_rollback** command to perform the rollback. This is because, unlike **mp** jobs, the multifile commands do not perform automatic recovery when they are rerun. The following shell script will roll back any outstanding multifile commands:

```
for recfile in *.rec
do
   if [ -f "$recfile" ]
   then m_rollback $recfile
fi
done
```

Recovery with multiple job steps

If the script contains multiple job steps (where a job step is an **mp run** command, a multifile command, or a UNIX command that changes the state of the file system), then it is necessary to guard against repeated execution of the same job step. This may be done by conditional logic in the script. For example:

```
if [[ ! -d /mfs1/my-dir ]]
then m_mkdir /mfs1/my-dir; fi
```

In summary

To configure a system for high-availability clustering, the following tasks must be completed:

- All data modified by Ab Initio must be placed on shared storage.
- All hosts must be referred to via service names or via the host alias file.
- Fail-over scripts must be set up to remap file system mount points, remap service names, set up a host alias file (if desired), and relocate information from the Ab Initio recovery directories (**/var/abinitio**).
- Restoration of service scripts must be set up to restore file system mount points, service names, the host alias file, and to either cleanup or relocate the contents of the recovery directories.
- Resource issues must be considered, particularly in light of possible overcommitment of main memory.
- Application-level recovery planning may be required for scripts that access databases, that contain multifile commands, or that contain multiple job steps.

Performance-tuning an IBM SP system

Ab Initio includes several features that improve scalability of program performance on large IBM SP configurations. These features are:

- The configuration variables AB_TCP_WRITE_BUFFER and AB_TCP_READ_BUFFER, which help in tuning the SP's high-speed switch.
- The option **-stages** to partition components and certain flows, which reduces kernel memory requirements on very large configurations (60 or more hosts).

Ab Initio configuration variables

The Ab Initio system transmits data over the SP's high-speed switch using TCP/IP. If the switch and TCP/IP options are not correctly configured, data flows that use the switch may have unexpectedly poor performance (say, about 300 KB/sec, rather than 20+ MB/sec that even a moderately tuned implementation can achieve).

This section presents a brief discussion of how TCP/IP works across the high-speed switch, of how the Ab Initio system uses TCP/IP, and of the circumstances under which performance problems may arise.

For more detailed descriptions of TCP/IP and how to tune the switch, consult the relevant IBM technical documentation, e.g., POWERparallel Service Bulletin No. 13 (http://www.rs6000.ibm.com/support/sp/sp_secure/bulletins) and the manual pages for the AIX no command.

How TCP/IP works

You can think of a TCP/IP connection as a pipe that can hold a finite amount of data in-transit (this is called the window). When the pipe fills up, the sender cannot send further data until the pipe empties. Each time this happens, performance is lost. In addition, each pipe consumes certain kernel-resident memory regions (the send and receive pools for the switch). If these memory regions are overcommitted, packets are lost and the TCP/IP subsystem has to retransmit them. Again, performance suffers.

To get good performance from TCP/IP, you want data to stream smoothly through this pipe. Several things can interrupt the streaming of the data:

- Certain network option settings can essentially disable streaming, leading to very poor performance (about 300 KB/sec per TCP/IP connection).
- If the windows are too small, the TCP/IP connection will stutter as the window repeatedly fills up and empties. Again, poor performance results.
- If the kernel memory pools are overcommitted, the kernel will sometimes have to discard data, forcing the sender to re-transmit it. This is invisible to the user, except for the poor performance that results. Under extreme circumstances, the kernel may get "stuck" and shut down a TCP/IP connection, leading to application failure.
- If individual data transmissions are too small, the overhead involved may result in mediocre performance (a few MB/sec). This situation is much less destructive than the other scenarios mentioned above.

Tuning the switch

The configuration variables AB_TCP_WRITE_BUFFER and AB_TCP_READ_BUFFER control the size of the window. The configuration variable AB_TCP_IO_SIZE controls the amount of data sent on each I/O.

If it appears that the kernel is dropping packets, try making AB_TCP_READ_BUFFER and AB_TCP_WRITE_BUFFER smaller. If it appears that data is not streaming through the pipes, either increase AB_TCP_READ_BUFFER and AB_TCP_WRITE_BUFFER or decrease AB_TCP_IO_SIZE.

Tuning the AIX networking subsystem

An alternative approach is to tune the AIX networking subsystem. Here are some brief notes on how to do this. Consult the relevant IBM documentation and/or IBM customer support for more detailed information.

The AIX **no** command may be used to examine and set certain parameters that govern the performance of TCP/IP. Here are the most important parameters:

rfc1323	Enables an extension to TCP/IP permitting window sizes greater than 64 KB. It is very important to set this to 1. If it is zero, it limits the TCP/IP pipe size to 64 KB. Since the switch tries to transmit data in 64-KB chunks, this will entirely prevent streaming, and almost guarantee poor performance.
sb_max	The maximum permitted window size on any connection (send + receive ends). This needs to be larger than the sum of *tcp_sendspace* and *tcp_recvspace*; otherwise results will be unpredictable.
tcp_sendspace	Default size, in bytes, of the sender window.
tcp_recvspace	Default size, in bytes, of the receive window. This parameter may have a variety of values - between **221184** and **655360** seems to be typical. Larger values decrease the probability that the pipes will "stutter", but increase the probability that kernel buffers will become overcommitted.

These parameters may be examined by using the **no -a** command at the shell prompt. Root may alter these parameters using the **no** command (consult IBM documentation). This does not require re-booting the system.

Tuning via pool size

IBM has recently added a tunable parameter for setting the size of the kernel buffers used to stage incoming/outgoing TCP/IP traffic. These are not set or examined by the **no** command. These parameters are called *spoolsize* (send pool size) and *rpoolsize* (receive pool size). To see their current values, type:

```
lsattr -E -l css0
```

The default value is **524288** bytes, but most installations seem to have them set to **2097152**. The default can be changed by using the **chgcss** command on each host then rebooting.

IBM recommends that the total *rpoolsize* be equal to the sum of *tcp_recvspace* for all TCP/IP connections coming into a host. For example, if *tcp_recvspace* is set to **221184** and a machine has **30** TCP/IP streams coming into a host, then the recommended rpoolsize would be **665320** (slightly over 6 MB). This would happen if the machine had an all-to-all or fan-in flow.

In practice, the Ab Initio system seems to work quite well when the rpoolsize is below this IBM-recommended size, but it is not clear how far below this limit you can safely go. If you are getting poor throughput on an all-to-all flow, or if you are getting excessive skew (one or more hosts falling behind), you should run **netstat -s** on the implicated hosts and check the number of retransmitted packets under the TCP statistics. If you are getting a lot of retransmission, then you should consider:

- Increasing the *rpoolsize*; or
- Decreasing the window sizes, via the AB_TCP_READ/WRITE_BUFFER configuration variable or via the **no** command. If you decrease the window sizes, you may then need to decease the I/O sizes (AB_TCP_IO_SIZE) to prevent stuttering.

Checking skew with AB_REPORT

The Ab Initio environment variable AB_REPORT can contain the keyword **skew**, which instructs the system to report data flow partitions that contain extremely high or extremely small amounts of data. In Release 1.2.10, the system will, by default, report the four worst partitions of each flow; this is to spare the user the potential for hundreds of skew reports in large configurations. To override this default of 4, use:

skew=*threshold***:***limit*

where threshold is the minimum skew-value to be reported, and limit is the maximum number of skew-values to be reported. If limit is set to all or *****, then all skewed partitions will be reported.

Multistage repartitioning

When data is repartitioned between an input layout and an output layout that both have a large number of data partitions, some operating system resources (e.g., TCP/IP buffers) may be unduly stressed. To reduce resource demands in such situations, Ab Initio provides multistage repartitioning, which allows you to choose to pay a small price in time in return for using fewer operating system resources.

Typically, a repartitioning step involves data flowing into a partitioning component, through an all-to-all flow, and then into a gather component. For *N* partitions, the all-to-all flow can require operating system resources proportional to *N***N*. With multistage repartitioning, the resource requirements are reduced to roughly **2****N****SQRT(***N***)**. For example, if a single-stage 25-to-25-way repartitioning requires 25*25 = 625 TCP/IP buffers, a two-stage repartitioning will require 2*25*5 = 250 TCP/IP buffers.

To use multistage repartitioning with **mp**, specify the number of stages desired as the argument to the **-stages** option to a partitioning component *and* the all-to-all flow it is connected to. Currently, the only values supported for **-stages** are 1 and 2.

- Specifying **-stages 1** gives one-stage repartitioning, which is the default behavior.
- Specifying **-stages 2** gives two-stage repartitioning.

For example, a two-stage hash repartitioner can be constructed as follows:

```
...
mp hash-partition  hasher  name  -layout  input  -stages  2
mp gather  gather  -layout output
mp all-to-all-flow  flow  hasher.out  gather.in  -stages  2
...
```

B>

Configuration variables

This appendix consists of a complete list of Co>Operating System configuration variables.

AB_AGENT_COUNT

Default value: 1

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: If set, controls the number of agents per machine. Specifically, if a machine is mentioned N times in a multifile system, the Co>Operating system spreads those N partitions across AB_AGENT_COUNT agents. This is done by generating host aliases on-the-fly when a multifile is expanded.

See also "AB_AGENT_COUNT sets number of agents on SMPs" on page 103.

AB_AGENT_ERROR_DELAY

Default value: 5

Type: integer

Group: error

Flags: hetero, wizard

Exported: true

Details: Sets the "grace period" between error detection and error processing. This parameter delays aborting a graph to give error messages, debugging output, and so on, time to make it to disk before the graph is aborted. In production, it may be advisable to set this to a fairly high value.

AB_AGENT_STARTUP_RETRY_COUNT

Default value: 2

Type: integer

Group: error

Flags: wizard

Exported: false

Details: Determines how many times the launcher will retry starting the agent on a particular machine. For example, a value of 2 (the default) results in up to 3 tries total, and a value of 0 results in only one try. Of course, the launcher will retry only upon failure.

AB_AGENT_UNITOOL_SERVER

Default value: -1

Type: integer

Group: tuning

Flags: wizard

Exported: true

Details: If non-negative, the minimum number of unitools for which an agent will start up a unitool server instead of launching the unitools individually. If negative, unitool server support is disabled. Not supported on Windows.

AB_ALLOW_DANGEROUS_KEY_CASTING

Default value: false

Type: bool

Group: compatibility

Flags: hetero, wizard

Exported: false

Details: Enables “dangerous” key casting for **merge-join** and **merge-transform**. In previous versions of the Co>Operating System, **merge-join** and **merge-transform** could operate with keys having different data types, for example **integer** and **decimal**. As of release 2.1, such “dangerous” mixtures of keys have been disallowed. This configuration variable re-allows such dangerous mixtures.

AB_ALLOW_GRAPH_IN_COMPONENT

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: true

Details: If true, allow the construction of graphs within components of other graphs. Such operations remain unsupported, however, and are likely to fail in mysterious ways.

See also "AB_ALLOW_GRAPH_IN_COMPONENT" on page 106.

AB_ALLOW_LITERAL_PREFIX

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, literal strings may have an **E** or **A** prefix, and non-native literals without a prefix may be shown with a prefix. E.g., on an **ascii** machine:

```
    ebcdic string("\n")
```

will be shown as

```
    ebcdic string(E"\045")
```

Unicode literal strings always have a **U** prefix.

AB_ALLOW_NFS_WORKDIR

Default value: false

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: If true, allow the use of an NFS-mounted working directory for the agent. This is not recommended because of the added network traffic and reduced performance.

AB_ALWAYS_SPILL

Default value: false

Type: bool

Group: tuning

Flags: hetero, wizard

Exported: false

Details: Forces the **hash-join**, **hash-aggregate**, **hash-rollup**, and **hash-scan** components to write their intermediate results to disk regardless of the value of their **-max-core** parameters.

AB_AUTO_MAX_CORE_POOL_SIZE

Default value: 256M

Type: string

Group: graph

Flags: common

Exported: false

Details: Specifies the total size of the pool of memory from which components using **-auto-max-core** will draw from.

AB_CACHEABLE_SYSTEM_DIRS

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: false

Details: This list of directories identifies areas which are eligible for file caching even if they exist in directory trees specified by AB_SYSTEM_DIRS. The following directory is always implicitly considered to be cacheable:

```
/~ab_home/config
```

AB_CATALOG

Default value: Unset

Type: url

Group: environment

Flags: common

Exported: false

Details: Defines the location of the lookup table catalog.

AB_CDL_DIR

Default value: /~ab_home/lib/layered_components

Type: url

Group: environment

Flags: common

Exported: false

Details: Location of component definition files for built-in components.

AB_CHARACTER_DISPLAY_ESCAPE

Default value: true

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Enables dumping of non-printable characters as escape sequences.

AB_CHARACTER_DISPLAY_ORIGINAL

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Enables printing of original user-provided string constants.

AB_CHECK_CLOSE

Default value: true

Type: bool

Group: tuning

Flags: wizard

Exported: true

Details: If false, ignore errors that occur as file descriptors are implicitly closed. Errors when files are explicitly closed are unaffected.

AB_CHECK_FOR_FILE_CONFLICTS

Default value: true

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: Check that in a single phase the same file is neither both read and written nor written twice.

AB_CHECK_HETERO_DATA_TYPES

Default value: true

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: Where flows cross from one host to another with different integer formats, floating-point formats, or character set, verify that these characteristics are completely specified in the metadata for the flow.

AB_COMPARE_EBCDIC_AS_ASCII

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: In releases before 2-8-18, DML string comparisons of EBCDIC operands used the ASCII collating sequence on an ASCII-based machine. After that release, EBCDIC collating order is used. Setting this flag to true allows the old behavior to be obtained. For example, consider the EBCDIC strings

```
estr1="0Aa "

estr2="aA0 "
```

The default behavior is EBCDIC collating. For example:

```
(estr1 > estr2) is 1.
```

But with the flag set to true,

```
(estr1 > estr2) is 0.
```

AB_COMPATIBILITY

Default value: Unset

Type: string

Group: compatibility

Flags: common

Exported: true

Details: Causes the Co>Operating System to behave compatibly with older releases. For example, between 2.0 and 2.1 there were changes in the handling of layouts for serial files. Setting AB_COMPATIBILITY to **2.0**, or **2.0.11**, and so on, retains the prior behavior.

See also "AB_COMPATIBILITY and emulation of previous releases" on page 107.

AB_COMPONENT_STDERR_KBYTES

Default value: 1000

Type: integer

Group: tuning

Flags: common

Exported: false

Details: Limits the amount of error output in Kbytes accepted from any one component before the component is considered to have failed without exiting. After reading this much error output, the agent stops reading and kills the babbling component.

AB_CONFIGURATION

Default value: Unset

Type: string

Group: environment

Flags: common

Exported: false

Details: A list of files where configuration variables may be specified. Separate items in the list with a colon (:) on UNIX platforms and with a semicolon (;) on Windows platforms. These files are read before ~/.abinitiorc or $AB_HOME/config/abinitiorc.

AB_CONNECTION

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Specifies the method to use for logging in to remote machines. Can be: **rsh**, **rexec**, **telnet**, **rlogin**, **ssh**, or **dcom**. Not all methods are available on all platforms.

AB_CONNECTION_EXTRA_ARGS

Default value: Unset

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Some of the connection methods involve executing programs that can take optional arguments. Normally these are not required, but if necessary this configvar can be used to set them. Arguments should be separated by spaces, and embedded spaces within a single argument are not supported.

AB_CONNECTION_SCRIPT

Default value: Unset

Type: string

Group: startup

Flags: hetero, wizard

Exported: false

Details: Specifies the script to be used for **rlogin** and **telnet**.

Defaults to /~**ab_home/lib/rlogin.script** for **rlogin**.

Defaults to /~**ab_home/lib/telnet.script** for **telnet**.

AB_CONNECTOR_REUSEADDR

Default value: true

Type: bool

Group: startup

Flags: hetero, wizard

Exported: false

Details: If set, TCP sockets to be used for connecting to socket listeners sets the SO_REUSEADDR option, encouraging the recycling of expiring port numbers. Since the address of the connecting socket is ephemeral, this should be safe on all platforms.

AB_COPY_USES_TICKETS

Default value: false

Type: bool

Group: tuning

Flags: hetero, wizard

Exported: true

Details: Because they do no significant computing, mostly I/O, copy components are exempt from the limit imposed by AB_TICKETS unless AB_COPY_USES_TICKETS is set.

AB_CUSTOM_MACRO_LIST

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: false

Details: Use this to register custom macros installed in the directory specified by AB_LAYERED_COMPONENTS_PATH. The custom macro components are specified with the **.mpm** extension. For example, if you have installed **test1.mpm**, **test1.usage**, **test2.mpm**, and **test2.usage** in the layered components directory, then set this variable to: **test1 test2**

AB_DATA_DIR

Default value: /~ab_work_dir/data

Type: url

Group: environment

Flags: hetero, common

Exported: false

Details: The directory for temporary data files that lack a specific, writable path in their layouts.

During job execution, components may need to write temporary data to disk. This can occur for large sorts, as a means of passing data across phase boundaries, for debugger files (see AB_EXPLICIT_MULTIFILE_DEFAULT_DIR), and so on. In most cases, temporary files are maintained in the .WORK subdirectory of a component's layout. But when you use host layouts or database layouts, which both lack directory paths, or when you use unwritable layout directories, AB_DATA_DIR provides disk storage for the temporary files.

This is set to **/~ab_work_dir/data** by default. The default is not ideal because the data files share the same filesystem as recovery files (see AB_WORK_DIR). The temporary data files are often large and can unexpectedly fill up their filesystem. Although the Co>Operating System can normally detect a filled filesystem and recover a job that fills it, if the temporary data files share the filesystem with the recovery files, the job could be unrecoverable. To avoid this situation, use a different filesystem for AB_DATA_DIR. One would generally use a filesystem with ample storage where regular layouts already appear. A RAID-5 filesystem is suitable for typical AB_DATA_DIR usage.

See also "AB_DATA_DIR specifies temporary file directory" on page 108.

AB_DEADLOCK_BUFFER_SIZE

Default value: 1048576

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Sets the amount of memory used per-process for deadlock buffering.

AB_DEBUG

Default value: Unset

Type: string

Group: debug

Flags: common

Exported: true

Details: Controls component debugging. Its value consists of one or more entries separated by colons (:), where an entry consists of:

component.partition

The *component* is the name of a component, as given in an MP script. The *partition* is a three-digit partition number. The wildcard character (*) may appear in the component, partition, or both.

AB_DECIMAL_ADD_COMPATIBILITY

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, decimal addition and subtraction use older algorithms that are less uniform in output formatting but are slightly faster.

AB_DECIMAL_CONSTANTS

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, constants of the form *ddd.ddd* are treated as decimal constants; if false, they are treated as floating-point constants.

AB_DECIMAL_FULL_INTERMEDIATE

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, intermediate decimal results are computed using temporaries that do not lose precision. Only multiply and divide are affected by this flag.

AB_DECIMAL_MULDIV_DOUBLE

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: Specifies how decimal multiplication and division are done. If set to true, use the old, less accurate way of multiplying and dividing decimals by converting to doubles, operating and converting back.

AB_DEFAULT_LOCAL_FLOW_TYPE

Default value: pipe

Type: string

Group: graph

Flags: wizard, hetero

Exported: false

Details: Sets the data transport method between components on the same machine. Possible values are:

tcp	TCP/IP sockets
usoc	UNIX domain sockets
pipe	Named pipes (the default)
mem	Shared memory

The flow type **usoc** (UNIX domain socket) is available for local flows (flows between components running on the same machine).

The **usoc** flows have lower overhead than TCP connections and offer better buffering than named pipes. Unfortunately, when the operating system's pool of buffer space for UNIX domain sockets is exhausted, the failure is abrupt and causes graph execution to fail. For this reason, **usoc** is not the default.

Please note that UNIX domain sockets rendezvous on local file system paths, not on port numbers. In a directory listing, a **usoc** looks like a named pipe with no permissions **(p---------)**. When you use a **usoc** in a graph, it appears in each agent's working directory with the suffix **.usoc**.

See also "AB_DEFAULT_LOCAL_FLOW_TYPE" on page 109.

AB_DEFER_FILE_SETUP

Default value: true

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: If true, defer creation of output files until the phase in which they're written; likewise defer checking for the existence of input files until the phase in which they're read. If false, these activities occur in the job's first phase no matter which phase the files are actually written or read. A possibly unwanted consequence of

```
AB_DEFER_FILE_SETUP=false
```

is that the creation of output files will be committed at the job's first checkpoint even if the files haven't yet been written to. A desirable consequence is that missing input files will be detected sooner than they might otherwise be in a multi-phase job.

AB_DIRECT_IO_THRESHOLD

Default value: -1

Type: integer

Group: tuning

Flags: hetero, wizard

Exported: false

Details: When reading data files over AB_DIRECT_IO_THRESHOLD megabytes on supported filesystems, direct I/O is used. For smaller files, sequential I/O is used. If AB_DIRECT_IO_THRESHOLD is negative, direct I/O is not used regardless of file size. When AB_DIRECT_IO_THRESHOLD is non-negative, direct I/O is used, where supported, for writing all data files. Direct I/O minimizes I/O cost by bypassing system-provided buffering and caching. It's most suitable in situations where the aggregate disk I/O of a job exceeds the system's caching capacity, effectively defeating the cache mechanism. Direct I/O is supported on Windows, and for local Solaris filesystems and for Veritas (vxfs) filesystems on Solaris and HP-UX.

AB_DISPLAY_CHARSET

Default value: ascii

Type: string

Group: environment

Flags: wizard

Exported: false

Details: Defines the character set used by the user’s display. At present, this only affects the **m_dump** command.

AB_DISTRIBUTE_BINARIES

Default value: false

Type: bool

Group: graph

Flags: wizard, hetero

Exported: false

Details: Enables distribution of binary executables to one or more machines. Ordinarily, the file cache mechanism distributes executables only if they are known to be shell scripts. This is because the mechanism does not know what operating system is running on the processing machine, and therefore has no reason to assume that a particular binary works on a particular machine.

AB_DML_VERSION_1

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: false

Details: Causes DML to be processed as in Release 1.3 and earlier.

AB_DONT_OPTIMIZE_LOCAL_LAUNCH

Default value: false

Type: bool

Group: startup

Flags: wizard

Exported: false

Details: Disables local-launching by forking. Normally, if the Co>Operating System starts up an agent locally, the startup is done by forking a new process. If this variable is set to true, the system uses the method specified by AB_CONNECTION.

AB_DUMP_RAW_CHARACTERS

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Unprintable characters are dumped in unescaped mode.

AB_EBCDIC_PAGE

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: true

Details: Uses one of the following values to define the default EBCDIC code page for conversions between EBCDIC and ASCII data:

ebcdic_page_1047
ebcdic_page_500
ebcdic_page_037
ebcdic_page_modern

If not set, **ebcdic_page_modern** is used. This is similar to **ebcdic_page_500** except that the mapping of **newline** and **linefeed** is reversed. MVS files often use **ebcdic_page_modern**, or **ebcdic_page_500** but Unix System Services uses **ebcdic_page_1047** for conversions.

AB_ENABLE_KEY_CONCATENATION

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, enables an optimization whereby adjacent key fields may be treated as a single key field for the purposes of comparisons.

AB_ENABLE_VFILES

Default value: false

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: Enables the use of vfiles (vector-files) for checkpoint files. You should enable vfiles if the amount of data saved in checkpoint files exceeds 2 Gbytes and the platform lacks large-file support. Most platforms now support large files.

AB_ENCRYPT_FLOWS

Default value: false

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: Activates encryption on all TCP-based data flows.

AB_ENCRYPTED_PASSWORD

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Supplies a password for one or more processing nodes. The password is stored in an encrypted form. Use the **m_password** utility to set this in an initialization file.

AB_ENHANCED_LOOKUP

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: Permits enhanced lookup functionality, including multiple records in the table having the same key value.

AB_ERROR_STRING_DIR

Default value: Unset

Type: url

Group: environment

Flags: wizard

Exported: true

Details: Directory containing error string catalogs. If not set, the directory is $AB_HOME/lib.

AB_EXPLICIT_MULTIFILE_DEFAULT_DIR

Default value: /~ab_data_dir/debugger

Type: string

Group: environment

Flags: common

Exported: false

Details: The **m_mkfile -newfiles** command creates data partitions in the specified directory when no paths are provided for the data partitions. If undefined, the directory **/~ab_data_dir/debugger** is used. Note that the debugger uses this to create temporary files for flows that have host or database layouts (see AB_DATA_DIR).

AB_EXPORT

Default value: Unset

Type: string

Group: environment

Flags: hetero, wizard

Exported: false

Details: Lists the configuration variables whose values you want the control machine to export to processing machines. By default, the control machine always exports certain configuration variables. To find out whether a configuration variable is exported by default, see the description for the configuration variable. Separate successive entries with a colon (:). For example

```
var1:var2:var3
```

AB_FILE_IO_SIZE

Default value: 65536

Type: integer

Group: tuning

Flags: common

Exported: true

Details: Sets the default I/O size in bytes for file-based data flows when memory-mapping is not in use.

See also "AB_IO_SIZE and other configuration variables set size of I/O operations" on page 110.

AB_FILECACHE_DISABLE

Default value: false

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: Turns off file caching.

AB_FILECACHE_HOSTNAME

Default value: Unset

Type: string

Group: graph

Flags: wizard

Exported: false

Details: Forces a specific name to use as the filecache host. Except in the simplest of cases (when there is only one processing node involved), one would typically define this as a heterogenous configuration variable in a form such as:

AB_FILECACHE_HOSTNAME @ *old-name* **:** *new-name*

AB_FILECACHE_TRACE

Default value: false

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: Enables tracing of the file cache. When this is turned on, all filecache activity will be logged to **stdout**.

AB_FIXED_TYPE_FOR_CONSTANT_STRINGS

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, this configuration variable causes string constants to be given fixed size types. If false, they have delimited types.

AB_FLOW_ERROR_DELAY_FACTOR

Default value: 3

Type: integer

Group: error

Flags: hetero, wizard

Exported: false

Details: Many errors in reading or writing flows are caused by the failure of connected components. Reporting of these secondary errors is delayed by the product of AB_FLOW_ERROR_DELAY_FACTOR and AB_AGENT_ERROR_DELAY so that the agent has a chance to report the primary error and clean up.

AB_FLOWFILE_CREATION_MODE

Default value: u=rw

Type: string

Group: tuning

Flags: common

Exported: false

Details: Specifies the permissions for a newly created flow file or pipe in either symbolic format, such as *ug=rwx*, or octal numeric format, such as *0664*. The default permissions allow access only to the user who runs the graph, thereby preventing other users from removing files after graph failure.

AB_FORBIDDEN_TCP_LISTENER_PORTS

Default value: 1521

Type: string

Group: environment

Flags: hetero, common

Exported: true

Details: A list of TCP ports that the Co>Operating System is not to use for TCP socket listeners. Separate successive entries with a colon (:). The default value, **1521**, is the "well-known" but unreserved port used for Oracle's TNS listener process.

AB_FOREGROUND_SORT

Default value: false

Type: bool

Group: tuning

Flags: hetero, wizard

Exported: false

Details: Determines whether sorting is done by a background process.

AB_FSYNC_LOG

Default value: true

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: Ensures synchronous writes to log files. If this variable is set to false, writes to log files are not synchronous. This somewhat reduces job-startup time but may result in recovery failures in the event of a machine crash.

AB_FTP_DEBUG

Default value: false

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: Turns on tracing of FTP commands for the FTP component.

AB_FTP_MAX_RETRY_COUNT

Default value: 10

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: If AB_FTP_RETRY_OK_CODE is true, how many attempts the **ftp-from** component should make to perform the file transfer. Multiply this by the value of AB_FTP_RETRY_PAUSE_SECONDS to determine the time that elapses before **ftp-from** gives up.

AB_FTP_PORT

Default value: 0

Type: integer

Group: startup

Flags: hetero, wizard

Exported: true

Details: If greater than zero, specifies the connection port for FTP. If zero, the standard or locally-configured FTP port is used. Use heterogeneous configuration variable settings such as the following to set ports on a per-machine basis:

AB_FTP_PORT @ *host*

See also "AB_TELNET_PORT and AB_FTP_PORT specify non-standard ports" on page 114 and "AB_IO_SIZE and other configuration variables set size of I/O operations" on page 110.

AB_FTP_REMOTE_LINE_BREAK

Default value: Unset

Type: string

Group: tuning

Flags: wizard

Exported: false

Details: This setting is needed (and only used) when attempting to restart a failed ASCII connection with an FTP server that does not support block transfer mode. If not set, then such connections are not restarted. Legal values are NL or CRNL depending on whether the remote FTP server uses newline (as in Unix) or carriage return/newline to represent end-of-line in ASCII files.

AB_FTP_RESTART_ATTEMPT_INTERVAL_MSEC

Default value: 1000

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: The length of time that the FTP component pauses between attempts to restart a broken connection.

AB_FTP_RESTART_MAX_ATTEMPTS

Default value: 2

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: The maximum number of times the FTP component attempts to restart a broken connection without having made some progress since the last attempt.

AB_FTP_RETRY_OK_CODE

Default value: false

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: Normally when the **ftp-from** component issues a **get** (RETR) command it expects to receive code 226 from the remote FTP server to indicate success, and any other code is treated as a failure. But if AB_FTP_RETRY_OK_CODE is set to true then the **ftp-from** component treats values other than 226 as transient errors if (and only if) no bytes have yet been obtained from the remote file. The number of attempts and the time between attempts is configurable with AB_FTP_MAX_RETRY_COUNT and AB_FTP_RETRY_PAUSE_SECONDS.

AB_FTP_RETRY_PAUSE_SECONDS

Default value: 5

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: If AB_FTP_RETRY_OK_CODE is true, how long the **ftp-from** component should sleep before retrying the file transfer. Multiply this by the value of AB_FTP_MAX_RETRY_COUNT to determine the time that elapses before **ftp-from** gives up.

AB_GETHOSTBYADDR_RETRY_COUNT

Default value: 5

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Number of times to retry gethostbyaddr in xxi_gethostbyname. If AB_GETHOSTBYXXXX_ALWAYS_RETRY is not set, the retries are done only on timeout-related errors.

AB_GETHOSTBYNAME_RETRY_COUNT

Default value: 5

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Number of times to retry gethostbyname in xxi_gethostbyname. If AB_GETHOSTBYXXXX_ALWAYS_RETRY is not set, the retries are done only on timeout-related errors.

AB_GETHOSTBYXXXX_ALWAYS_RETRY

Default value: false

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: If this is true, the retry loops of gethostbyname and gethostbyaddr in xxi_gethostbyname will retry even if the error is authoritatively an error. If it is false, the default, it only retries on timeout-type errors.

AB_GLOBAL_PACKAGE

Default value: /~abinitio/global_package

Type: string

Group: environment

Flags: wizard

Exported: false

Details: Defines the "global package" in use.

AB_HIDE_ENV_VAR

Default value: Unset

Type: string

Group: environment

Flags: hetero, wizard

Exported: false

Details: Lists the environment variables whose names and values should be hidden when capturing the process or job environment for later display. Separate successive entries with a colon (:). For example

```
    var1:var2:var3
```

AB_HOME

Default value: Unset

Type: url

Group: environment

Flags: hetero, common

Exported: false

Details: The directory where the Co>Operating System is installed. If the control-machine directory is the same as on the processing machines, it is not necessary to set this variable for those machines.

When you execute Co>Operating System commands from the command-line, this must be set in the environment rather than in a configuration file.

Usual value on Unix platforms:

```
/usr/local/abinitio
```

Usual value on Windows platforms:

```
C:\AbInitio
```

AB_HOST_ALIAS_FILE

Default value: Unset

Type: url

Group: environment

Flags: common

Exported: false

Details: A file that contains logical-to-physical host mappings.

The host alias file has a series of entries of the form:

logical_machine physical_machine

This causes the logical machine to be mapped, at runtime, to the specified physical host.

AB_HOST_INTERFACE

Default value: Unset

Type: url

Group: environment

Flags: common

Exported: false

Details: Controls the network interface used for control-machine-to-processing-machine communication. If a host (the control machine) has multiple interfaces, then it is usually a good idea to set AB_HOST_INTERFACE to refer to the fastest interface. For example, if the host has an ethernet interface and a high-speed switch, then AB_HOST_INTERFACE should generally refer to the switch. All machines used in the computation must be able to communicate with the interface that is specified. If they cannot, the job fails at startup-time.

AB_HOSTNAME

Default value: Unset

Type: string

Group: environment

Flags: wizard, hetero

Exported: false

Details: Establishes a "primary hostname" for systems having multiple interfaces. This lets the graph compiler have a better notion of which components are on the same machine.

AB_HPUX_IPC_SHARE32

Default value: true

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: On 64-bit HP-UX, create shared memory segments so that they can be shared with 32-bit processes. Set this to false to enable the larger 64-bit memory pool, but only when using a pure 64-bit build.

AB_HPUX_MEMORY_WINDOW

Default value: Unset

Type: string

Group: tuning

Flags: wizard

Exported: false

Details: Put all Co>Operating System processes into a memory window. The value can be a window name (first column of **/etc/services.window)** or a window ID (second column of **/etc/services.window**). HP-UX only.

AB_IFILE_DIRECT_IO_KBYTES

Default value: 0

Type: integer

Group: tuning

Flags: common

Exported: false

Details: On some operating systems, direct I/O enables components to bypass the operating system's file system cache. A value of zero disables direct I/O. With a positive value, most read operations will be direct and to a buffer of that size. If this value is positive, the value of AB_IFILE_MMAP_KBYTES must be zero. Direct I/O is supported only on Windows, Solaris, and HP-UX.

AB_IFILE_MMAP_KBYTES

Default value: 4096

Type: integer

Group: tuning

Flags: common

Exported: false

Details: Sets the default memory-mapped buffer size of input files in Kbytes. A zero value disables the use of memory-mapping for input files. For debugging purposes, a negative value disables the usual fallback to normal I/O if the file cannot be mapped. This value is ignored, and Memory-mapping is never used for input files, on Windows, HP-UX, and Linux.

AB_IGNORE_FUNCTION_CACHE

Default value: false

Type: bool

Group: error

Flags: wizard

Exported: false

Details: Causes function evaluation to act as if the function result cache (**function_value_value** rel) does not exist.

AB_INCLUDE_FILES

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: false

Details: A list of DML files that DML code should implicitly include. Separate files in the list with a colon (:) on UNIX platforms and with a semicolon (;) on Windows platforms.

AB_INCLUDE_FILES_ARE_PRIVATE

Default value: true

Type: bool

Group: compatibility

Flags: wizard

Exported: false

Details: Causes included DML files to be loaded into a private context rather than into the shared global context.

AB_INLINE_TRANSFORM_FUNCTIONS

Default value: true

Type: bool

Group: environment

Flags: wizard

Exported: true

Details: Controls inline substitution of simple transform functions.

AB_IS_INTERNATIONAL

Default value: Unset

Type: boolean

Group: environment

Flags: common

Exported: false

Details: A value of "true" enables Unicode for the internals of the Co>Operating System and EME, allowing DML names and file names to contain any Unicode character and allowing error messages in languages other than English.

This variable does not affect the processing of data; even with a value of "false", DML can manipulate data in any supported character set.

AB_JOB

Default value: Unset

Type: string

Group: graph

Flags: common

Exported: false

Details: Specifies the base name of the job file for the current job. This overrides the use of the file **.abinitio-current-job** that would otherwise identify the current job file. It provides an implicit argument to an **mp job** command that is invoked with no **jobname** argument. Use this variable to differentiate two jobs that run at the same time in the same directory.

AB_LAYERED_COMPONENTS_PATH

Default value: /~ab_home/lib/layered_components

Type: path

Group: environment

Flags: wizard

Exported: false

Details: A list of directories where the Co>Operating System looks for **.mpc** files. Separate items in the list with a colon (:) on UNIX platforms and with a semicolon (;) on Windows platforms.

AB_LAYOUT_EXACT

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: false

Details: Changes the default for **mp layout** to be **exact** rather than **workdir**.

AB_LAYOUT_WORK_DIR

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: false

Details: Causes serial-file layouts to have their temporary files in **/~ab_data_dir**. As of 2.1, the default behavior is to put temporary files in a **.WORK** directory.

AB_LIB_PATH

Default value: Unset

Type: path

Group: environment

Flags: hetero, common

Exported: false

Details: List of directories to be searched for libraries of user-defined extensions to DML and compiled transforms. The directories must be absolute pathnames. Separate items in the list with a colon (:) on UNIX platforms and with a semicolon (;) on Windows platforms.

AB_LISTENER_BACKLOG

Default value: 50

Type: integer

Group: tuning

Flags: wizard, hetero

Exported: false

Details: Sets the pending-connection backlog for sockets. For the meaning of the backlog, see the UNIX system documentation for the **listen** system call.

AB_LISTENER_REUSEADDR

Default value: false

Type: bool

Group: startup

Flags: hetero, wizard

Exported: false

Details: If set, TCP sockets to be used for creating socket listeners set the SO_REUSEADDR option, encouraging the recycling of expiring port numbers. A bug in AIX versions 4.1.4 through 4.2.x (IBM APAR IX69864) makes this otherwise good idea dangerous, but it should be safe on other platforms.

AB_LOCAL_NAMES

Default value: Unset

Type: string

Group: tuning

Flags: wizard

Exported: false

Details: When it is set, it is a colon-separated list of machine names and IP addresses that all refer to the local machine. It is currently only used on supported Windows platforms. It must never be exported and normally should not be set by the user. The variable is supposed to be self-initializing. The variable exists to ease IP lookup performance problems that can occur when running on a machine that has been disconnected from its network.

AB_LOCAL_NETRC

Default value: Unset

Type: url

Group: startup

Flags: wizard

Exported: false

Details: Specifies an alternative **.netrc** file to look at for username and password information. If unset, **~/.netrc** is read on UNIX. On Windows, **netrc** or **.netrc** is read from **$SystemRoot/system32/drivers/etc**. If set to an empty string or to a non-existent file, no file is read and no error is reported. If set to an existing file, the file must exist and either be zero-sized or have permissions so it can only be read by the owner.

AB_LOCALS_NON_NULL

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: Causes transform function failure if a local variable is assigned null.

AB_LOCK_FILES

Default value: true

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: Switch that bypasses file locking for operations that typically perform file locking. WARNING: Setting AB_LOCK_FILES to false bypasses normal file checking and can result in errors during graph runs. Such errors are known to occur specifically in graphs using Continuous Flows components.

AB_LOG_FILE

Default value: Unset

Type: url

Group: error

Flags: wizard

Exported: false

Details: Location where Co>Operating System errors may be logged.

AB_LOG_TIMES

Default value: false

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: Instructs some components to record user, system, and wall-clock times for certain internal phases and to emit the times on the log port.

AB_MAX_DATA_PARALLELISM

Default value: 0

Type: integer

Group: graph

Flags: wizard

Exported: false

Details: Limits the data parallelism of any component. Extra phase breaks are inserted as necessary to reduce parallelism. The default value, **0**, turns off this feature.

AB_MAX_RECORD_BUFFER

Default value: 5000000

Type: integer

Group: error

Flags: wizard

Exported: true

Details: This variable defines the largest record that the Co>Operating System processes. The purpose of this variable is to cause the Co>Operating System to abort when an implausibly large record is found. This situation generally arises when there is a mismatch between the data and the metadata for variable-length records, for example, an error in specifying the delimiter for a string. If a very large record is legitimately encountered, the value of this variable may be increased to suppress the resulting abort.

AB_MAX_RPC_QUEUE_BYTES

Default value: 4194304

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Controls the maximum size in bytes of pending RPCs before the queue is flushed.

AB_MAX_SKEW

Default value: Unset

Type: integer

Group: monitoring

Flags: wizard

Exported: false

Details: Defines the threshold for reporting multifile skew.

AB_MAX_TRACE_FILE_SIZE

Default value: 100000000

Type: integer

Group: tuning

Flags: wizard

Exported: true

Details: This is the size limit for trace files produced by **XXtrace**. Any time the trace file exceeds this size, the first half of the file is thrown away. This configuration variable may ONLY be set in the environment. It may not be set in a configuration file.

See also "AB_MAX_TRACE_FILE_SIZE sets size limit for trace files" on page 111.

AB_MEM_IO_BUFFERS

Default value: 3

Type: integer

Group: tuning

Flags: common

Exported: true

Details: Sets the number of buffers (of AB_MEM_IO_SIZE each) allocated for each memory-based flow.

AB_MEM_IO_SIZE

Default value: 65536

Type: integer

Group: tuning

Flags: common

Exported: true

Details: Sets the default I/O size in bytes for memory-based data flows.

AB_METADATA_TRANSLATORS_PATH

Default value: /~ab_home/lib/metadata_translators

Type: path

Group: environment

Flags: wizard

Exported: false

Details: Defines the location of metadata translators.

AB_MF_VERBOSE_ERROR_KEYWORDS

Default value: inconsistency:data partition

Type: string

Group: environment

Flags: wizard

Exported: false

Details: Multifile commands, such as **m_ls**, **m_rm**, and **m_touch**, report errors tersely by default. This is a list of keywords which, if they appear in the detailed report, cause the detailed report to be shown in place of the terse one. Separate keywords in the list with a colon (:). To get verbose reporting of all errors, include the keyword **verbose** in your AB_REPORT setting.

AB_MMAP_MAX_WAIT

Default value: 60

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Maximum amount of time, in seconds, to wait for an **mmap** to succeed.

AB_MP_CHECK_ALWAYS

Default value: false

Type: bool

Group: error

Flags: wizard

Exported: false

Details: By default, the graph command interpreter, **mp**, defers error-checking the **mp** commands in a deployed script until the **mp run** command. If set to true, the configuration variable causes **mp** to check each command for errors. This checking drastically slows graph startup.

AB_MSYNC_RETRIES

Default value: 9

Type: integer

Group: tuning

Flags: hetero, wizard

Exported: false

Details: If calling **msync(2)** fails with EAGAIN as the reported error, the Co>Operating System retries **msync** AB_MSYNC_RETRIES times with an increasing delay between retries, governed by AB_MSYNC_RETRY_MSECS.

AB_MSYNC_RETRY_MSECS

Default value: 500

Type: integer

Group: tuning

Flags: hetero, wizard

Exported: false

Details: If calling **msync(2)** fails with EAGAIN as the reported error, the Co>Operating System retries **msync** AB_MSYNC_RETRIES times with a delay of (retry_number) * AB_MSYNC_RETRY_MSECS milliseconds between retries.

AB_MV_INTO_DIRECTORY

Default value: true

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: If true, **m_mv** moves the source file or source directory into an existing target directory as does the UNIX **mv** command. Before Release 2.9, **m_mv**replaced the existing target directory.

If AB_COMPATIBILITY is set to a release before Release 2.9 and you want **m_mv** to move the source into the target directory instead of replacing it, you must set AB_MV_INTO_DIRECTORY to true explicitly.

AB_NICE

Default value: Unset

Type: integer

Group: tuning

Flags: hetero, common

Exported: true

Details: Determines the value for the UNIX **nice** parameter. Higher values mean lower priorities. This may be used to reduce the priority of Co>Operating System components, which is helpful in preventing components from swamping the CPU and preventing other jobs from getting any processing time. The maximum legal value is **20**.

AB_NO_AUTO_ROLLBACK

Default value: false

Type: bool

Group: debug

Flags: common

Exported: false

Details: Suppresses automatic rollback-on-error. This is useful if you need to check the state of the file system at the moment an error was detected.

AB_NO_DEFAULT_CONFIGURATION

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Suppresses loading of the default configuration files. This is ignored by Windows platforms. This variable must be set in the environment. Its setting in a configuration file is ignored.

AB_NODE_KILL_TIMEOUT

Default value: 10

Type: integer

Group: error

Flags: hetero, wizard

Exported: false

Details: Controls the number of seconds the Co>Operating System waits for components to exit. When a job is aborted, the Co>Operating System sends all components the TERM signal, waits for a number of seconds, then sends any remaining components the KILL signal. The TERM signal can be "caught" by the application, allowing it to perform some action before exiting. If the cleanup action takes too long, however, the KILL signal, which cannot be caught, kills the program before the cleanup is completed. Extending the node **kill** timeout gives the component more time.

AB_NODES

Default value: Unset

Type: string

Group: environment

Flags: hetero, common

Exported: false

Details: AB_NODES establishes names for groups of hostnames. These names can then be used to keep an abinitiorc file relatively short and readable. The general form is

```
AB_NODES @ groupname : name1 name2 name3 ...
```

A setting which applies to all of the named hosts can then be written as

AB_FOO @ groupname : setting

See $AB_HOME/config/abinitiorc.example for more examples.

AB_NPIPE_IO_SIZE

Default value: 0

Type: integer

Group: tuning

Flags: common

Exported: true

Details: Sets the default I/O size in bytes for FIFO-based data flows. If this is zero, the system's own FIFO buffer size is used on UNIX platforms and AB_WIN32_PIPE_BUFFER is used on Windows platforms.

See also "AB_IO_SIZE and other configuration variables set size of I/O operations" on page 110.

AB_NPIPE_READER_OPEN_DELAY

Default value: 10

Type: integer

Group: error

Flags: wizard, hetero

Exported: true

Details: The number of seconds the reader should wait before opening a named pipe attached from a custom component. This is set to lessen synchronization errors under which named pipe readers can hang if the writer opens a named pipe, writes 0 bytes, and closes it. This is ignored under Solaris, which implements named pipe opening correctly.

AB_NPIPE_WRITER_CLOSE_DELAY

Default value: 10

Type: integer

Group: error

Flags: wizard, hetero

Exported: true

Details: The number of seconds the writer should wait before closing a named pipe attached to a custom component if nothing has been written into the pipe. This is set to lessen synchronization errors under which named pipe readers can hang if the writer opens a named pipe, writes 0 bytes, and closes it. This is ignored under Solaris, which implements named pipe opening correctly.

AB_NUMA_Q

Default value: -1

Type: integer

Group: tuning

Flags: common

Exported: false

Details: On Sequent NUMA-Q systems, agents and the components they spawn are tied to particular quads (processor clusters) if this is non-zero, otherwise they run on whatever quads the O/S assigns them to. If AB_NUMA_Q is positive, the number of agents is the product of AB_NUMA_Q and AB_AGENT_COUNT. If AB_NUMA_Q is negative, the actual number of quads is substituted.

AB_NUMA_Q_MMAP_PROT_EXEC

Default value: true

Type: bool

Group: tuning

Flags: common

Exported: false

Details: On Sequent NUMA-Q systems, memory-mapped files are not usually replicated on different quads (processor clusters) even if they are read-only. Memory-mapping with PROT_EXEC allows replication of the pages for improved performance.

AB_NUMA_Q_PRIME

Default value: -1

Type: integer

Group: tuning

Flags: common

Exported: false

Details: On Sequent NUMA-Q systems when AB_NUMA_Q is non-zero, this determines which quad is considered the first one when distributing agents among the available quads (processor clusters). Any negative value indicates the quad on which the driver program was launched. If AB_NUMA_Q is **1**, the entire graph runs on the quad specified by AB_NUMA_Q_PRIME.

AB_NUMBER_OF_PARTITIONS

Default value: Unset

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Gives the number of partitions in a component's layout. The Co>Operating System sets this in the environment of a component. It allows a component to find out how many partitions it has. See also AB_PARTITION_INDEX.

AB_OFILE_DIRECT_IO_KBYTES

Default value: 0

Type: integer

Group: tuning

Flags: common

Exported: false

Details: On some operating systems, direct I/O enables components to bypass the file system cache when writing to files. A value of zero disables direct I/O upon write. With a positive value, most write operations will be direct and from a buffer of that size. If this value is positive, the value of AB_OFILE_MMAP_KBYTES must be zero. Direct I/O is supported only on Windows, Solaris, and HP-UX.

AB_OFILE_MMAP_KBYTES

Default value: 0

Type: integer

Group: tuning

Flags: common

Exported: false

Details: Sets the default memory-mapped buffer size of output files in Kbytes. A zero value disables the use of memory-mapping for output files. For debugging purposes, a negative value disables the usual fallback to normal I/O if the file cannot be mapped. This value is ignored, and Memory-mapping is never used for output files, on Windows, HP-UX, and Linux.

AB_OLD_ASCII_MAP

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: Causes ASCII-to-EBCDIC conversion to treat ASCII as a 7-bit character set.

AB_OLD_DECIMAL_DECIMAL_CONVERSIONS

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, this configuration variable causes decimal-to-decimal conversion to drop insignificant zeroes.

AB_OLD_DOUBLE_DECIMAL_CONVERSIONS

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: Specifies how reals are converted to decimals and vice versa. Slight differences may result due to differences in the algorithms used.

AB_OLD_STAR_TRANSLATION

Default value: false

Type: bool

Group: compatibility

Flags: wizard

Exported: true

Details: If true, expands star (*) assignments the old way, as multiple value assignments.

If true

```
out.* :: in.*;
```

is expanded the old way, as

```
out.a, out.b, out.c :: in.a, in.c, in.c;
```

If false

```
out.* :: in.*;
```

is expanded as

```
out.a :: in.a;

out.b :: in.b;

out.c :: in.c;
```

AB_OPTIMIZE_TRANSFORM_FUNCTIONS

Default value: true

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Controls optimization of transform functions.

AB_PACKED_DECIMAL_IS_STRICT

Default value: true

Type: bool

Group:

Flags:

Exported:

Details: the range of the sign nibble in packed decimal data is checked. The degree of checking is governed by the setting of the configuration variable AB_PACKED_DECIMAL_IS_STRICT, as follows:

- If the value of the configuration variable is 0, the range of sign nibbles in packed decimal data is not checked.
- If the value is 1, the value of the sign nibble in packed decimal data must be greater than or equal to 0xa.

- If the value is 2 (this is the default setting beginning in Release 2.10.7), the value of the sign nibble must be 0xc, 0xd, or 0xf.

  For example, the string `E"  "` (EBCDIC space, or 0x404040) would be invalid under this default setting .

AB_PARTITION_INDEX

Default value: Unset

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Gives the index of a component's partition. The Co>Operating System sets this in the environment of a component. It allows a component to find out which partition it is. See also AB_NUMBER_OF_PARTITIONS.

AB_PASSPHRASE

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Supplies a pass phrase for remote connections via the "secure shell client" program **ssh**. This configuration variable does not normally need to be set. The only circumstance when it might be useful is when AB_CONNECTION is set to **ssh** and the "secure shell client" expects a pass phrase rather than a password. Even in that case the value of AB_PASSWORD is used by the Co>Operating System for the pass phrase if this configuration variable is not set.

AB_PASSWORD

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Supplies a password for one or more processing machines. If you supply this parameter, it should be stored in a file that only you can read. If it is stored as an environment variable or in a file that other users can read, then security may be compromised.

AB_PROFILE_DIRECTORY

Default value: Unset

Type: url

Group: environment

Flags: wizard

Exported: false

Details: An absolute pathname of a directory for recording profiling data from components. Each component's vertex name (e.g., "sort.000") is used as a filename within the directory to avoid name collisions. Profiling is disabled if AB_PROFILE_PATH is unset or zero-length.

AB_PROTOCOL_CHECK

Default value: true

Type: bool

Group: tuning

Flags: wizard

Exported: false

Details: If true, verify that the driver on the control machine is protocol-compatible with the agents on the processing machines.

AB_PSTACK

Default value: false

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: If set, emit a stack trace before aborting for any reason. Available only on Sun Solaris, IBM AIX, Hewlett-Packard HP-UX, Silicon Graphics IRIX, Red Hat Linux, Windows, Compaq True64 UNIX, and NCR MP-RAS.

AB_PSTACK_AGENT

Default value: true

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: If set, emit a stack trace before aborting the agent. Available only on Sun Solaris, IBM AIX, Hewlett-Packard HP-UX, Silicon Graphics IRIX, Red Hat Linux, Windows, Compaq True64 UNIX, and NCR MP-RAS.

AB_PSTACK_COMPONENTS

Default value: false

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: If set, emit a stack trace before aborting a built-in component, including **unitool**, **adaptor**, **generate**, **table-bcast**, and **transform**. These components produce a stack trace on deadly signals (SIGSEGV, SIGBUS, etc.) in any case. Available only on Sun Solaris, IBM AIX, Hewlett-Packard HP-UX, Silicon Graphics IRIX, Red Hat Linux, Windows, Compaq True64 UNIX, and NCR MP-RAS.

AB_RANDOM_SEED

Default value: normal

Type: string

Group: tuning

Flags: common

Exported: true

Details: Specification of how to seed the random number generator. The default value, **normal**, seeds the random number generator within each component with a value computed from the component name and partition number. If set to **random** the seed is computed from the component name, partition number, and current time. If set to any other value, the seed is computed from the component name, partition number, and that value.

AB_REMAP_FD_TYPES

Default value: all

Type: string

Group: tuning

Flags: wizard

Exported: true

Details: Specifies the types of file descriptors to attempt to remap. File descriptors will be remapped only on those operating systems where buffered IO can access only 256 file descriptors. Allowed values are:

• none	Do not remap fds
• file	Remap fds for files
• npipe	Remap fds for named pipes
• all	Remap fds for both files and named pipes

AB_REMOTE_EXECUTION_PROTOCOL_VERSION

Default value: Unset

Type: string

Group: compatibility

Flags: wizard

Exported: true

Details: Controls the version of the launcher-agent-component protocol which is used during remote execution. During normal execution, the launcher detects the version of each remote agent and sets this variable appropriately. Setting it externally provides an upper bound; the launcher may still select an older version if that's necessary, but it will never select a newer version.

AB_REPORT

Default value:

Type: string

Group: monitoring

Flags: common

Exported: true

Details: Controls monitoring and error reporting. Contains a series of entries, separated by spaces. Here are the possible entries:

verbose	Turns on verbose error reporting.
monitor, flows	Turns on flow monitoring.
monitor=*n*	Turns on flow monitoring, with reports at *n*-second intervals.
interval=*n*	Show reports at *n*-second intervals. See also AB_REPORT_INTERVAL. The default is 1 second.
table-flows	Turns monitoring on for the flows used to distribute lookup tables.
processes, times	Monitor process CPU usage.
split-cpu	For process monitoring, provides separate columns for user and system CPU usage.
spillage	For process monitoring, provides additional **Maxcore** and **Spilled Bytes/Records** columns to report max-core usage and the amount of data spilled to disk. For flow monitoring, provides an additional **Buffer** column to report bytes spilled for flow buffering. For flow buffers, a K, M, or G is appended to indicate kilobytes, megabytes, or gigabytes.
totals	For process monitoring, provides an additional line labeled **Total** that shows total CPU usage and spillage for all processes.
skew	Shows up to 4 flow partitions with skew whose absolute value exceeds 25%.
skew=*n[***:***value]*	Shows flow partitions with skew whose absolute value exceeds the threshold *n*. The optional :*value*, which is 4 by default, is the maximum number of partitions to show. The value *all* or * shows all partitions whose skew exceeds the threshold.

scroll=*true/false*	Determines whether report output is scrolled or overwritten in the terminal window. The default is to overwrite.
file=*path*	Causes all terminal-window reporting to be instead written to a file.
summary=*path*	Causes a summary report to be written to a file.
file-percentages	For flow monitoring, provides a percentage-done **(%)** column to show progress of file-input flows.
error-codes	Displays alphanumeric error codes with error messages.

See also "AB_REPORT" on page 112.

AB_REPORT_INTERVAL

Default value: -1

Type: integer

Group: monitoring

Flags: wizard

Exported: false

Details: Controls the frequency of monitoring reports.

AB_RETAIN_LOGS

Default value: auto

Type: string

Group: debug

Flags: wizard

Exported: false

Details: Controls whether recovery information is retained.

Values are:

yes	Always retain recovery logs
no	Never retain recovery logs
auto	Delete recovery logs if job succeeds

AB_REXEC_CHARSET

Default value: Unset

Type: string

Group: startup

Flags: wizard, hetero

Exported: false

Details: Overrides the default startup method of inferring the character set of a processing machine from its output when using **rexec**. Valid values are **ascii** and **ebcdic**.

AB_REXEC_PORT

Default value: Unset

Type: integer

Group: startup

Flags: wizard, hetero

Exported: false

Details: Specifies a port to use for **rexec** connections rather than the one returned by looking up the **rexec**service.

AB_RLOGIN

Default value: rlogin

Type: url

Group: startup

Flags: wizard

Exported: false

Details: Specifies the name of the “remote login” program. The default is the **rlogin** program found in the user path.

AB_RSH

Default value: C:\WINDOWS\System32\rsh.exe

Type: url

Group: startup

Flags: wizard

Exported: false

Details: Specifies the name of the “remote shell” program. On various systems, it is called **rsh**, **remsh**, or **rsh.exe**. The default value depends on the platform as follows:

HP-UX	**/bin/remsh**
DYNIX/ptx	**/usr/bin/resh**
IRIX	**/usr/bsd/rsh**
Windows	**/C:\WINNT\System32\rsh.exe**
z/OS	**/usr/local/bin/rsh**
Other platforms	**/usr/bin/rsh**

AB_RTEL_ABORT_STRINGS

Default value: incorrect:Incorrect:invalid:Invalid

Type: string

Group: startup

Flags: hetero, wizard

Exported: false

Details: A list of strings separated by colons (:). When any of them are encountered during the process of establishing a remote login using either the **telnet** or **rlogin** connection method, the login attempt immediately aborts and reports an error. These strings typically appear in the error messages from **telnet** (and **rlogin**) daemons when the login has failed for some reason. But at some sites the words in the default string might appear in strictly informational messages, and at those sites the values for this configuration variable should be changed to omit the innocuous values. The consequence of missing a string that really is reporting an error is that the connection failure won’t be detected until the AB_RTEL_TIMEOUT_SECONDS interval has expired.

AB_RTEL_NAME_PROMPT

Default value: login:|Username:|Login:|Name:

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Specifies the possible spellings for the user name prompt when making **rlogin**/**telnet** connections to a processing machine. Separate alternatives with a vertical bar (|).

AB_RTEL_PASSWORD_PROMPT

Default value: Password:|password:

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Specifies the possible spellings for the user password prompt when making **rlogin**/**telnet** connections to a processing machine. Separate alternatives with vertical bar (|).

AB_RTEL_PAUSE_MSECS

Default value: 200

Type: integer

Group: startup

Flags: wizard

Exported: false

Details: Milliseconds to wait before typing part of a command line to **telnet**/**rlogin**.

AB_RTEL_PROMPT

Default value: %|#|$|>

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Specifies the shell prompt expected by **rlogin**/**telnet** on a processing machine. Separate alternatives with vertical bar (|).

AB_RTEL_SHELL

Default value: sh

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Specifies the shell run on the processing machine when making **rlogin**/**telnet** connections. Possible values are **sh**, **ksh**, **csh** and **bat**.

AB_RTEL_TERM

Default value: dumb

Type: string

Group: startup

Flags: wizard

Exported: false

Details: Sets the terminal type for **telnet** sessions during startup.

AB_RTEL_TIMEOUT_SECONDS

Default value: 120

Type: integer

Group: startup

Flags: wizard

Exported: false

Details: Controls how long to wait for **rlogin**/**telnet** to respond.

AB_RTEL_USE_TELNET

Default value: true

Type: bool

Group: startup

Flags: hetero, wizard

Exported: false

Details: Tells the Co>Operating System whether to use **telnet** rather than **rlogin** for remote startup.

AB_RTEL_VERBOSE

Default value: 0

Type: integer

Group: startup

Flags: wizard

Exported: false

Details: Controls verbosity of **telnet** connect tracing.

AB_RUN_EXTRA_PATH

Default value: ../..

Type: string

Group: environment

Flags: common

Exported: true

Details: When a graph component launches controlled child programs, it is generally convenient if the PATH environment variable can be expanded so that they can be easily found. This allows the user to turn off this feature.

AB_SCREEN_EXPORTS

Default value: false

Type: bool

Group: tuning

Flags: common

Exported: false

Details: If true, verify that exported configuration variables contain only printable characters. Unprintable characters, such as control characters or non-ASCII characters on ASCII systems, are likely to cause problems with remote job startup.

AB_SEMMSL

Default value: 4

Type: integer

Group: error

Flags: wizard

Exported: false

Details: Controls the number of semaphores allocated in each semaphore array.

AB_SEND_RETRIES

Default value: 9

Type: integer

Group: tuning

Flags: hetero, wizard

Exported: false

Details: If writing to a **tcp** or **unix** domain socket fails with ENOBUFS, ENOMEM, or ENETDOWN as the reported error, the write is retried AB_SEND_RETRIES times with an increasing delay (governed by AB_SEND_RETRY_MSECS) between retries.

AB_SEND_RETRY_MSECS

Default value: 500

Type: integer

Group: tuning

Flags: hetero, wizard

Exported: false

Details: If writing to a **tcp** or **unix** domain socket fails with ENOBUFS, ENOMEM, or ENETDOWN as the reported error, the write is retried AB_SEND_RETRIES times with a delay of (retry number) * AB_SEND_RETRY_MSECS milliseconds between retries.

AB_SEQUENT_EFS_EXTENT_SIZE

Default value: 16

Type: integer

Group: tuning

Flags: common

Exported: false

Details: On Sequent EFS file systems, set the default number of blocks per extent for growing files. A value of zero means to use the system default.

AB_SERVER_RETRY_ATTEMPTS

Default value: 5

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: This is relevant for Windows platforms only. This value controls the number of times that a process trying to start another process through the Ab Initio Service attempts to open the named pipe for communication. There is a one-second sleep between attempts.

AB_SERVER_UMASK

Default value: -1

Type: int

Group: debug

Flags: wizard

Exported: false

Details: If set, this is the umask to use when creating files with the Ab Initio server.

AB_SHMEM_BYTES

Default value: 2000000

Type: integer

Group: error

Flags: common

Exported: true

Details: Controls the size in bytes of the Co>Operating System shared memory region. In unusual circumstances, the shared memory region may overflow. Increasing this value makes the shared memory region larger, effectively working around the overflow. See also AB_SHMEM_DUMP_FILE.

AB_SHMEM_DUMP_FILE

Default value: Unset

Type: url

Group: debug

Flags: wizard

Exported: true

Details: If the shared memory space specified by AB_SHMEM_BYTES is exhausted, dump information about the current contents into the specified file.

AB_SHMEM_INITIAL_BUCKETS

Default value: 256

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: Governs the initial number of buckets in the Co>Operating System's shared-memory hash table.

AB_SHOW_COMPILED_GRAPH

Default value: false

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: Causes the graph compiler to show compiled graphs on **stderr**.

AB_SKEW_THRESHOLD

Default value: 999

Type: integer

Group: monitoring

Flags: wizard

Exported: false

Details: Determines the smallest skew that is reported in monitoring output. The default of **999** means that no skew is reported, since the largest possible skew value is **100**.

AB_SORT_PROCESSES

Default value: 1

Type: integer

Group: tuning

Flags: wizard

Exported: false

Details: The default number of background sort processes to use.

AB_SSH

Default value: ssh

Type: url

Group: startup

Flags: wizard

Exported: false

Details: Specifies the name of the "secure shell client" program. The default is to take the **ssh** program found in the user's path. This connection method is not available on MP/RAS and Numa-Q platforms.

AB_SSH_NEEDS_PASSWORD

Default value: true

Type: bool

Group: startup

Flags: wizard

Exported: false

Details: Specifies whether the "secure shell client" program prompts for a password (or pass phrase) or not. If this value is true and the connection method is **ssh**, then AB_PASSWORD or AB_PASSPHRASE generally also needs to be set. This connection method is not available on MP/RAS and Numa-Q platforms.

AB_STARTUP_TIMEOUT

Default value: 30

Type: integer

Group: startup

Flags: wizard

Exported: false

Details: Specifies number of seconds before timing out when starting processes on a processing machine. To start a job on a processing machine, the Co>Operating System uses **rsh**, **rexec**, **rlogin**, **telnet**, **ssh**, or **dcom**. In most cases, these succeed or fail within a few seconds. However, if the processing machine is overloaded, startup may take significantly longer. Increasing this timeout gives the processing machine more time to respond.

AB_STRUCTURED_ERRORS

Default value: false

Type: bool

Group: monitoring

Flags: wizard

Exported: true

Details: If this variable is undefined or false, the Co>Operating System emits all error messages in text format. If this variable is true, the system emits context-specific error messages in machine-readable format.

See also “AB_STRUCTURED_ERRORS” on page 113.

AB_SUBSCRIBE_CHECKPOINT_INTERVAL

Default value: 300

Type: integer

Group: tuning

Flags: common

Exported: false

Details: Controls how often a subscriber checkpoints when using time-based checkpointing. Set this to the desired number of seconds between checkpoints.

AB_SUBSCRIBE_WAIT_FOR_COMMIT

Default value: false

Type: bool

Group: tuning

Flags: wizard

Exported: true

Details: Instructs all subscribers to wait for checkpoints to commit before continuing processing.

AB_SUPPRESS_SIGPIPE

Default value: true

Type: bool

Group: error

Flags: common

Exported: false

Details: Suppresses the killing of components by SIGPIPE. The offending I/O operation fails with EPIPE instead, which is reported through the usual mechanisms.

Reporting of socket and pipe errors is delayed by the product of:

```
AB_AGENT_ERROR_DELAY * AB_FLOW_ERROR_DELAY_FACTOR
```

The delay allows the agent to hear first from the failing component, if there is one, and terminate other EPIPE recipients before they create large numbers of unnecessary reports. If the EPIPE really is the first sign of trouble, it is reported normally.

AB_SUPRESS_CONFIG_WARNINGS

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: false

Details: Suppresses warnings about config file errors. This variable must be set in the environment. Its setting in a configuration file is ignored.

AB_SYSTEM_DIRS

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: false

Details: A list of directories, such as **/bin**, on the machine where the Co>Operating System software is installed. Separate items in the list with a colon (:) on UNIX platforms and with a semicolon (;) on Windows platforms. On UNIX, the following directories are always implicitly considered system directories:

```
/bin:/usr/bin:/etc:/usr/local:/usr/ucb:$AB_HOME
```

AB_TABLE_BCAST

Default value: true

Type: bool

Group: graph

Flags: wizard

Exported: false

Details: When true, use the **table-bcast** component to distribute the contents of lookup tables to remote components that use those tables. When false, connect the tables to the remote components through adapters. In either case, local components read or memory-map the tables directly from disk. The default value of true generally results in better performance by broadcasting the needed tables synchronously, thereby reducing contention.

AB_TABLE_BLOCK_SIZE

Default value: 8000

Type: integer

Group: graph

Flags: wizard

Exported: false

Details: Controls the block size for lookup table broadcasting.

AB_TCP_CONNECTION_TOKEN

Default value: enabled

Type: string

Group: startup

Flags: wizard

Exported: true

Details: The location of the key that is used to verify all TCP connections to improve security. If set to an empty string, all additional connection security is disabled.

AB_TCP_IO_SIZE

Default value: 16384

Type: integer

Group: tuning

Flags: common

Exported: true

Details: Sets the default I/O size in bytes for TCP-based data flows.

See also "AB_IO_SIZE and other configuration variables set size of I/O operations" on page 110.

AB_TCP_KEEPALIVE

Default value: 1

Type: integer

Group: startup

Flags: hetero, wizard

Exported: false

Details: Controls the use of the SO_KEEPALIVE option on TCP sockets. SO_KEEPALIVE makes it more likely that the crash or disconnection of a processing machine is noticed promptly, but adds some overhead to TCP connections. The following are allowable values:

0	no SO_KEEPALIVE
1	only on control connections (default)
2	on all connections, both control and data

AB_TCP_LOCAL_BUFFER

Default value: Unset

Type: integer

Group: tuning

Flags: hetero, common

Exported: true

Details: Sets the TCP/IP send and receive buffer sizes in bytes for local connections.

AB_TCP_NODELAY

Default value: true

Type: bool

Group: tuning

Flags: hetero, wizard

Exported: false

Details: When true, sets TCP_NODELAY option on TCP data connections. This avoids the buffering delays inserted by some implementations to allow for coalescing successive packets.

AB_TCP_PORT_LEAST

Default value: 0

Type: int

Group: startup

Flags: hetero, wizard

Exported: false

Details: If AB_TCP_PORT_LEAST and AB_TCP_PORT_MOST are both non-zero, with MOST greater than LEAST, all TCP/IP sockets are bound to addresses from LEAST through MOST, inclusive.

AB_TCP_PORT_MOST

Default value: 0

Type: int

Group: startup

Flags: hetero, wizard

Exported: false

Details: If AB_TCP_PORT_LEAST and AB_TCP_PORT_MOST are both non-zero, with MOST greater than LEAST, all TCP/IP sockets are bound to addresses from LEAST through MOST, inclusive.

AB_TCP_READ_BUFFER

Default value: Unset

Type: integer

Group: tuning

Flags: hetero, common

Exported: true

Details: Sets the TCP/IP receive buffer size in bytes for remote connections.

AB_TCP_WRITE_BUFFER

Default value: Unset

Type: integer

Group: tuning

Flags: hetero, common

Exported: true

Details: Sets the TCP/IP send buffer size in bytes for remote connections.

AB_TELNET_PORT

Default value: 0

Type: integer

Group: startup

Flags: hetero, wizard

Exported: false

Details: If non-zero, specifies the connection port for **telnet**. If zero, the standard or locally-configured telnet port is used. Use heterogeneous settings such as the following to set ports on a per-host basis:

AB_TELNET_PORT@*host*

See also “AB_TELNET_PORT and AB_FTP_PORT specify non-standard ports” on page 114 and “AB_IO_SIZE and other configuration variables set size of I/O operations” on page 110.

AB_TELNET_TERMTYPE

Default value: dumb

Type: string

Group: startup

Flags: hetero, wizard

Exported: false

Details: Specifies the terminal type to be used for a **telnet** connection. Use heterogeneous settings such as the following to set ports on a per-host basis:

AB_TELNET_TERMTYPE@*host*

AB_TELNET_TRACE

Default value: false

Type: bool

Group: startup

Flags: wizard

Exported: false

Details: Turns on tracing of the telnet connection method.

AB_TICKETS

Default value: 0

Type: integer

Group: tuning

Flags: hetero, common

Exported: true

Details: If non-zero, limits the number of components per machine that are allowed to compute concurrently. I/O operations are excluded from this limit, as are copy components unless AB_COPY_USES_TICKETS is set. This configuration variable must be set to 0 on Windows platforms.

AB_TIMEOUT

Default value: 600

Type: integer

Group: error

Flags: wizard

Exported: false

Details: Controls how long to wait before timing out. This applies to most host-machine-processing-machine communications. The timeout value may be raised if jobs are failing due to slow communication between the control machine and the processing machines.

AB_TMPDIR

Default value: Unset

Type: string

Group: tuning

Flags: wizard

Exported: false

Details: A location to use instead of the temporary directory of the operating system.

If not set:

On non-Windows platforms, the Co>Operating System uses /tmp.

On Windows platforms, the Co>Operating System uses the value taken from the %TMP% or %TEMP% environment variables, in that order, if set, or from %SystemRoot%, which is C:\WINNT.

Note that this configuration variable may be set ONLY in the environment. It must *not* be set in a configuration file.

AB_UI_CHARSET

Default value: Unset

Type: string

Group: environment

Flags: hetero, common

Exported: true

Details: The character set (text encoding) used for user interface, such as standard I/O and command arguments. If the value is not present, the character set is assumed to be iso-latin-1 / Windows code page 1252 for ASCII based hosts such as UNIX machines and Windows NT / 2000, and EBCDIC for mainframes. The valid values for AB_UI_CHARSET are: “ascii”, “ebcdic”, “utf8”. “utf8” is only available for internationalized version of Ab Initio.

AB_UI_LANGUAGE

Default value: Unset

Type: string

Group: environment

Flags: hetero, common

Exported: true

Details: The variable designates the user interface language of Ab Initio commands. The default value is “en_US”, US English. Please consult the documentation for internationlized version of Ab Initio in order to find the value for particular langauge. For Japanese, it is “ja_JP”, for French, it is “fr_FR”.

AB_UI_LOCALE

Default value: Unset

Type: string

Group: environment

Flags: hetero, common

Exported: true

Details: The variable designates the locale of the user. The default value is “en_US”, US English. Please consult the documentation for internationlized version of Ab Initio in order to find the value for particular langauge. For Japanese, it is “ja_JP”, for french, it is “fr_FR”.

AB_ULIMIT

Default value: nofiles=max file=max data=max memory=max

Type: string

Group: tuning

Flags: hetero, common

Exported: true

Details: This configuration variable controls soft resource limit settings on Unix and z/OS USS, as in the **ulimit**command. These limits apply to all processes run as part of the job, even those on other machines and even third-party programs run as custom components. This configuration variable is ignored on Windows.

The value of the configuration variable must be a sequence of zero or more whitespace-separated clauses of the following form:

resource=limit

The *resource* is any of the following words. Each is a maximum per process:

coredump	Size in 512-byte blocks of a core dump file
data	Kilobytes of data area
file	Size in 512-byte blocks of any file written
nofiles	Number of file descriptors
stack	Kilobytes of stack area
time	CPU seconds
memory	Kilobytes of (virtual) memory
vmemory	Same as **memory**

The *limit* is either a number or one of the following words:

max	Same as the hard limit for this resource
unlimited	Remove the limit

For example:

```
nofiles=max data=20000 time=unlimited
```

AB_UNLINK_PIPES

Default value: true

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: Unlink named pipes used in soc-to-soc flows immediately after they are open at both ends. This removes them from the file system, saving on imachine updates as the pipes are used.

AB_USE_RTEL

Default value: false

Type: bool

Group: startup

Flags: hetero, wizard

Exported: false

Details: Tells the Co>Operating System to use **rlogin**/**telnet** for remote startup.

AB_USE_SHORT_NPIPE_FLOWNAMES_ON_WINDOWS

Default value: false

Type: bool

Group: graph

Flags: wizard

Exported: true

Details: By default, on Windows platforms, named pipe flows are created in the root of the named pipe namespace. Some applications have limits on the length of these names, which can grow quite large depending on the name of the component. Setting this configuration variable to true causes a sequence number to be used in place of the component name. This shortens the pipe name. You should not set this to true as a general rule because the shortened flow names can make flow analysis difficult or impossible.

AB_USE_STORAGE_POOLS

Default value: true

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: Controls the use of storage pools. You might want to disable this when searching for memory errors.

AB_USERNAME

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Gives the user name for remote login. If AB_USERNAME is different from the account name running the graph, then the behavior of the connection methods that do not normally specify a user name are altered.

rsh	Uses the -l option for rsh connections
exec	For local jobs, uses a remote connection method with the new user name

AB_VALIDATE_HOSTNAMES

Default value: true

Type: bool

Group: debug

Flags: wizard

Exported: false

Details: If set, machine names are validated before any call to **gethostbyname**. Each component of a valid machine name consists only of alphanumeric characters or a dash (-), but cannot begin or end with a dash. Components are separated by a period (.). No other characters are permitted.

AB_VERBOSE_FUNCTION_STATUS

Default value: false

Type: bool

Group: error

Flags: wizard

Exported: false

Details: Makes function evaluation return a detailed error status instead of just **novalue**.

AB_VERIFY_HOST_INTERFACE

Default value: true

Type: bool

Group: environment

Flags: common

Exported: true

Details: If true, verifies that AB_HOST_INTERFACE, if set, is a configured interface of the current host system. If false, accepts the setting of AB_HOST_INTERFACE without question.

AB_VFILE_MAX_SEGMENT_SIZE

Default value: Unset

Type: integer

Group: debug

Flags: wizard

Exported: false

Details: Determines the segment size for a new vfile when the segment size is not given. If not set, uses a default value of 1 Gbyte.

AB_WINDOWS_DOMAIN

Default value: Unset

Type: string

Group: startup

Flags: hetero, common

Exported: false

Details: Set this variable for remote login to Windows if both of the following are true:

(1) You are logging in as a Windows domain user, and

(2) You are using the MKS **rexec** daemon or another connection daemon that expects domain names.

Logging in as a domain user contrasts with logging in as a “local user,” which is known only to a specific machine. You can log into the specific machine as a user that is local to that machine without having to specify a domain.

MKS 8.0 or later requires that the domain name be specified. The only reason not to set this configuration variable is if the remote connection daemon does not allow it, such as Ataman.

The configuration variable AB_USERNAME must not include the domain. For example, to specify the Windows domain user **CAMELOT\Guinevere**, set the following:

```
AB_WINDOWS_DOMAIN @ windows_node : CAMELOT

AB_USERNAME @ windows_node : Guinevere
```

AB_WORK_DIR

Default value: /var/abinitio

Type: url

Group: environment

Flags: hetero, common

Exported: false

Details: The directory for files used by the Co>Operating System at run-time. This includes three sub-directories:

host	Holds control-machine recovery files.
vnode	Holds processing-machine recovery files.
cache	Holds caching files needed by remote components.

Because this directory contains recovery files, it is crucial that its filesystem never fill up. If it were to fill up during a job, the job could be unrecoverable. Ab Initio recommends allocating at least 500 Mbytes for this directory.

The subdirectories typically contain lots of small files that are subject to many small read and write operations. A RAID-1 (mirrored) filesystem provides optimal performance for these files.

See also AB_DATA_DIR.

AB_WORKDIR_CREATION_MODE

Default value: a=rwxt

Type: string

Group: tuning

Flags: common

Exported: false

Details: Specifies the permissions for a newly created .WORK directory in either symbolic format, such as *ug=rwx*, or octal numeric format, such as *0755*.

AB_XFR_ADVANCED_OPTIMIZATION_LOG_DIRECTORY

Default value: Unset

Type: url

Group: environment

Flags: wizard

Exported: true

Details: Target of logs from advanced optimization output.

AB_XFR_ADVANCED_OPTIMIZATION_LOG_LEVEL

Default value: 0

Type: int

Group: environment

Flags: wizard

Exported: true

Details: Controls level of logging advanced optimization of transform functions:

0	No logging
1	Output IR graphs and executable dumps
2	Turn on logging in optimization phases

AB_XFR_ADVANCED_OPTIMIZATION_OPTIONS

Default value: none

Type: string

Group: environment

Flags: wizard

Exported: true

Details: Controls level of advanced optimization of transform functions, colon delimited: (e.g. cfg:cse:cp:dc)

"none"	Do not perform advanced optimizations
"cfg"	Build control flow graph
"cse"	Common subexpression elimination
"cp"	Copy propagation
"dc"	Dead code elimination

AB_XFR_AST_CODEGEN

Default value: false

Type: bool

Group: environment

Flags: wizard

Exported: true

Details: Enables C++ code generation from a reconstructed AST.

AB_XFR_CODEGEN_DIRECTORY

Default value: Unset

Type: url

Group: environment

Flags: wizard

Exported: false

Details: Specifies the directory in which to place files for transform compilation.

AB_XFR_OPTIMIZATION_LEVEL

Default value: 2

Type: int

Group: environment

Flags: wizard

Exported: true

Details: Controls level of optimization of transform functions:

0	No optimization.
1	Remove useless moves.
2	Clean up branches and remove dead code.
3	Remove redundant tests and global variable refreshes.
4	Fold together moves of adjacent fields.
99	Do it all.

AB_XFR_PROFILE_LEVEL

Default value: Unset

Type: string

Group: environment

Flags: wizard

Exported: true

Details: Specifies how, if at all, transform functions should be profiled. If set to **function**, function-level information is collected. If set to **statement**, statement-level information is collected. If set to **compiled**, information about calls to compiled forms of functions is collected.

The profiling information for a component is output in a log record when a component has finished processing its data.

See also "AB_XFR_PROFILE_LEVEL and transformation engine profiling" on page 115.

Index