IBM InfoSphere DataStage
Version 11 Release 3

Programmer's Guide

SC19-4285-00

IBM InfoSphere DataStage
Version 11 Release 3

Programmer's Guide

SC19-4285-00

Note

Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 139.

Contents

Chapter 1. Command line interface

The InfoSphere® DataStage® CLI comprises four groups of commands, one for running jobs, one for administering projects, one for importing objects, and one for checking and repairing objects.

Commands for controlling InfoSphere DataStage jobs

You can start and stop jobs, and retrieve information about job runs by using the dsjob command.

The command options used with the dsjob command give you access to the same functionality as the InfoSphere DataStage API functions described in "API Functions" on page 38 or the BASIC functions described in "InfoSphere DataStage BASIC Interface" on page 95.

There is a single command, `dsjob`, with a large range of options. These options are described in the following topics:

- The logon clause
- Starting a job
- Stopping a job
- Listing projects, jobs, stages, links, and parameters
- Setting an alias for a job
- Retrieving information
- Accessing log files
- Generating a report

All output from the `dsjob` command is in plain text without column headings on lists, or any other sort of description. This enables the command to be used in shell or batch scripts without extra processing.

The InfoSphere DataStage CLI returns a completion code of 0 to the operating system upon successful execution, or one of the InfoSphere DataStage API error codes on failure. See "Error Codes" on page 90. The return code is also printed to the standard error stream in all cases. On UNIX servers, a code of 255 is returned if the error code is negative or greater than 254, to see the "real" return code in these cases, capture and process the standard error stream.

Note that, on UNIX servers, DSJOB has a runtime dependency on the libvmdsapi.so shared library.

Securing credentials and parameter values in `dsjob` commands

You can encrypt data and store the encrypted values in files to use in **`dsjob`** commands. To enter credentials in the command line, you can alternatively use credential prompting to hide your password in the command window.

About this task

You can use encrypted credentials and encrypted parameter file values in your **dsjob** commands so that you can avoid typing clear data on the screen when you run commands in the command line. See the *Encrypt command* documentation in the IBM® InfoSphere Information Server Administration Guide for information about this command.

Procedure

1. Encrypt your information by running the **encrypt** command, and copy and save the encrypted values in a file:
   - If you are encrypting credentials, create the credentials file (`*.txt`) and securely store the file. See the topic on the credentials file for restrictions and sample contents for the credentials file.
   - If you are encrypting job parameter values, store the encrypted values in the appropriate job parameters file by copying the encrypted output and pasting the value in the job parameter file. For example:

     ```
     job_parameter_name=encrypted_job_parameter_value
     ```

     You can create aliases for the job parameter values. You cannot encrypt job parameter names.

   Save the file.
2. Run the **dsjob** command. You can enter your credentials securely by using one of these methods:
   - **Using the credentials file.** If you are running a **dsjob** command that requires your user credentials, run your command with the **-authfile** parameter and specify the full path of the credentials file that you want to use. A sample syntax for using the **-authfile** parameter in a **dsjob** command that uses a parameters file with encrypted data is as follows:

     ```
     dsjob -authfile c:\cred_file.txt
      -run -paramfile c:\paramfile.txt dstage1 testJob
     ```
   - **Credentials prompting.** If you want to specify your credential data through the command line, specify only the **-url** or the **-domain** parameter, and the **-server** parameter. You are prompted for the user name and password. (The password is hidden in the command window.) If you include the **-url** or the**-domain** parameter, and the **-server** and **-user** parameters in your command, then you are prompted for the password. Here is a sample command and interaction:

     ```
     C:\IBM\InformationServer\Clients\Classic>dsjob
     -domain [2002:920:c000:217:9:32:217:32]:9443 -server RemoteServer
     -ljobs newTest
     Please type user name:admin
     Please type password:
     Job_ODBC
     Job_UDT5
     Job_UDT6
     Job_Universe
     PXJob_DC
     Sequence_ODBC
     Sequence_UDT5
     Sequence_UDT6
     Sequence_Universe

     Status code = 0
     ```

     Note: The password is hidden in the command window.

The logon clause

By default, the InfoSphere DataStage CLI connects to the engine on the local system using the user name and password of the user running the command.

For the **dsjob** and **dsadmin** commands, you can specify a different domain, engine, user name, or password using the logon clause, which is equivalent to the API **DSSetServerParams** function. Its syntax is as follows:

```
[-url domainURL |
 -domain domain_name ][ -user username ][ -password password ]
     [ -server enginename ]
```

or, for **dsjob** command only:

```
-domain NONE -user username -password password -server enginename
```

domainURL specifies a full format URL for the domain to log on to. The URL includes the protocol, host, and port information for the domain in this format: `https://domain:port`. The port defaults to 9443 if it is not specified.

domain_name specifies the domain to log on to. For **dsjob**, you can set `-domain NONE` to log on to the engine rather than the domain. In this case the user name and password are for the engine, not for the domain.

enginename specifies a different engine to log on to.

username specifies a different user name to use when logging on.

password specifies a different password to use when logging on.

Encrypted user names and passwords are not supported on the command line, use a credentials file if you want to use encrypted values.

If you do not want to type your credentials in the command line and you do not want to use a credentials file, specify only the **-url** or the **-domain** parameter, and the **-server** parameter, and you are prompted for the user name and password. (The password is hidden as you type in the command window.) If you include the **-url** or the **-domain** parameter, and the **-server** and **-user** parameters in your command, then you are prompted for the password. Here is a sample command and interaction:

```
C:\IBM\InformationServer\Clients\Classic>dsjob
-domain [2002:920:c000:217:9:32:217:32]:9443 -server RemoteServer
-ljobs newTest
Please type user name:admin
Please type password:
Job_ODBC

Status code = 0
```

For computers that do not use the default ports, to connect to a project that is on a local server, specify the **-server** option with only the port number and do not specify the server name. You do not need to specify your user name and password. For example:

```
dsjob -server :31539 -lprojects
```

For a more secure login for the **dsjob** and **dsadmin** commands, you can use a credentials file that can contain encrypted data:

```
-authfile credentials_filename
```

credentials_filename is the full path and name of the file that contains the logon details. This file supports encrypted and unencrypted data. See The credentials file for details and sample contents for the file.

You could alternatively specify the unencrypted details in a credentials file by using the following syntax:

-file *credentials_filename domainname enginename*

Note: The **-file** credentials file cannot contain encrypted data.

For the **dsjob** command, you can also use the command:

-file *credentials_filename* NONE *enginename*

domainname specifies the domain for which the file contains logon details. For **dsjob**, you can set NONE to log on to the engine rather than the domain. In this case the username and password are for the engine, not for the domain.

enginename specifies the engine for which the file contains logon details.

credentials_filename is the full path and name of the file that contains the logon details. This file supports only unencrypted data. The file should contain the following information if logging on to the domain:

domainname,enginename, *username*, *password*

The file must contain the following information if logging on to the engine:

enginename, *username*, *password*

Including the logon clause in your commands can expose your username and password. It is better to use the **-authfile** option and hold the encrypted logon information in a separate file, or to let the computer prompt you for your password.

Start a job

You can start, stop, validate, and reset jobs by using the -run option.

```
dsjob -run
[ -mode [ NORMAL | RESET | VALIDATE | RESTART ] ]
[ -param name=value ]
[ -paramfile filename ]
[ -warn n ]
[ -rows n ]
[ -wait ]
[ -stop ]
[ -jobstatus ]
[ -userstatus ]
[ -local ]
[ -opmetadata [ TRUE | FALSE ] ]
[ -disableprjhandler ]
[ -disablejobhandler ]
[ -queue ]
[ -useid ] project job | job_id
```

-mode

Specifies the type of job run. NORMAL starts a job run. RESET resets the job. VALIDATE validates the job. RESTART resumes a restartable job sequence from the last checkpoint by using the original job parameter values. If **-mode** is not specified, a normal job run is started.

-param
Specifies a parameter value to pass to the job. The value is in the format *name*=*value*, where *name* is the parameter name and *value* is the value to be set. If you use this to pass a value of an environment variable for a job (as you might do for parallel jobs), you need to use sing quotation marks with the environment variable and its value. For example, `-param '$APT_CONFIG_FILE=test.apt'`. Otherwise, the current value of the environment variable is used.

-paramfile
Specifies a file that contains the parameter values to pass to the job. The parameter values can be in the same format as **-param**, or values can be encrypted and stored in a parameter file as described in "Securing credentials and parameter values in **dsjob** commands" on page 1.

-warn *n*
Sets warning limits to the value specified by *n* (equivalent to the **DSSetJobLimit** function used with DSJ_LIMITWARN specified as the *LimitType* parameter).

-rows *n*
Sets row limits to the value specified by *n* (equivalent to the **DSSetJobLimit** function used with DSJ_LIMITROWS specified as the *LimitType* parameter).

-wait
Waits for the job to complete (equivalent to the **DSWaitForJob** function).

-stop
Stops a running job (equivalent to the **DSStopJob** function).

-jobstatus
Waits for the job to complete, then returns an exit code that is derived from the job status.

-userstatus
Waits for the job to complete, then returns an exit code that is derived from the user status if that status is defined. The user status is a string, and it is converted to an integer exit code. The exit code 0 indicates that the job completed without an error, but that the user status string was not converted. If a job returns a negative user status value, it is interpreted as an error.

-local
Use this when you are running a job from within a shell script on a UNIX system. Provided the script is run in the project directory, the job picks up the settings for any environment variables that are set in the script and any setting specific to the user environment.

-opmetadata
Use this to have the job generate operational metadata as it runs. If you specify TRUE, operational metadata is generated, whatever the default setting for the project. If you specify FALSE, the job does not generate operational metadata, whatever the default setting for the project.

-disableprjhandler
Use this to disable any error message handler that was set on a project-wide basis.

-disablejobhandler
Use this to disable any error message handler that was set for this job.

-useid
Specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

-queue
The name of the workload management queue that the job is submitted to. If you do not specify a queue, the project default queue is used. If the job is a sequence job, all jobs in the sequence job are submitted to this queue.

project The name of the project that contains the job.

job The name of the job. To run a job invocation, use the format *job.invocation_id*.

job_id An alias for the job that was set by using the **dsjob -jobid** command.

Stopping a job

You can stop a job using the **-stop** option.

```
dsjob -stop [-useid] project job|job_id
```

-stop terminates a running job (equivalent to the **DSStopJob** function).

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing the job.

job is the name of the job. To stop a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

Listing projects, jobs, stages, links, parameters, and queues

You can list projects, jobs, stages, links, job parameters, and workload management queues by using the **dsjob** command.

The different versions of the syntax are described in the following sections.

Listing projects

The following syntax displays a list of all known projects on the server:

```
dsjob -lprojects
```

This syntax is equivalent to the **DSGetProjectList** function.

Listing jobs

The following syntax displays a list of all jobs in the specified project:

```
dsjob -ljobs project
```

project is the name of the project containing the jobs to list.

This syntax is equivalent to the **DSGetProjectInfo** function.

Listing jobs with specific job statuses

The following syntax displays a list of all jobs in the specified project with specific job status values:

```
dsjob -ljobs [-status status_list] project
```

project is the name of the project containing the jobs to list.

status_list is a list of job status values as defined in the `dsapi.h` file.

The following command lists all jobs in the dstage1 project with statuses of DSJS_CRASHED or DSJS_STOPPED:

```
dsjob –ljobs -status 96/97 dstage1
```

Listing stages

The following syntax displays a list of all stages in a job:

```
dsjob -lstages [-useid] project job|job_id
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job containing the stages to list. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

This syntax is equivalent to the **DSGetJobInfo** function with DSJ_STAGELIST specified as the *InfoType* parameter.

Listing links

The following syntax displays a list of all the links to or from a stage:

```
dsjob -llinks [-useid] project job|job_id stage
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job containing *stage*. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

stage is the name of the stage containing the links to list.

This syntax is equivalent to the **DSGetStageInfo** function with DSJ_LINKLIST specified as the *InfoType* parameter.

Listing parameters

The following syntax display a list of all the parameters in a job and their values:

```
dsjob -lparams [-useid] project job|job_id
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job whose parameters are to be listed. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

This syntax is equivalent to the **DSGetJobInfo** function with DSJ_PARAMLIST specified as the *InfoType* parameter.

Listing invocations

The following syntax displays a list of the invocations of a job:

```
dsjob -linvocations [-useid] project job|job_id
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job whose parameters are to be listed. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

Listing workload management queues

The following syntax displays a list of the workload management queues:

```
dsjob -lqueues
```

Setting an alias for a job

The *dsjob* command can be used to specify your own ID for an InfoSphere DataStage job.

Other commands can then use that alias to refer to the job.

```
dsjob -jobid [my_ID] project job
```

my_ID is the alias you want to set for the job. If you omit *my_ID*, the command will return the current alias for the specified job. An alias must be unique within the project, if the alias already exists an error message is displayed.

project is the name of the project containing *job*.

job is the name of the job. To identify a job invocation, use the format *job.invocation_id*.

Retrieving information

The dsjob command can be used to retrieve and display the available information about specific projects, jobs, stages, or links.

The different versions of the syntax are described in the following sections.

Displaying job information

The following syntax displays the available information about a specified job:

dsjob -jobinfo [**-useid**] *project job|job_id*

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

The following information is displayed:

- The current status of the job
- The name of any controlling job for the job
- The date and time when the job started
- The wave number of the last or current run (internal InfoSphere DataStage reference number)
- User status

This syntax is equivalent to the **DSGetJobInfo** function.

Displaying stage information

The following syntax displays all the available information about a stage:

dsjob -stageinfo [**-useid**] *project job|job_id stage*

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job containing *stage*. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

stage is the name of the stage.

The following information is displayed:

- The last error message reported from any link to or from the stage
- The stage type name, for example, Transformer or Aggregator
- The primary links input row number

This syntax is equivalent to the **DSGetStageInfo** function.

Displaying link information

The following syntax displays information about a specified link to or from a stage:

```
dsjob -linkinfo [-useid] project job|job_id stage link
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job containing *stage*. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command (see "Setting an alias for a job" on page 8).

stage is the name of the stage containing *link*.

link is the name of the stage.

The following information is displayed:

- The last error message reported by the link
- The number of rows that have passed down a link

This syntax is equivalent to the **DSGetLinkInfo** function.

Displaying parameter information

This syntax displays information about the specified parameter:

```
dsjob -paraminfo [-useid] project job|job_id param
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job containing *parameter*. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

parameter is the name of the parameter.

The following information is displayed:

- The parameter type
- The parameter value
- Help text for the parameter that was provided by the job's designer
- Whether the value should be prompted for
- The default value that was specified by the job's designer
- Any list of values

- The list of values provided by the job's designer

This syntax is equivalent to the **DSGetParamInfo** function.

Accessing log files

The `dsjob` command can be used to add entries to a job's log file, or retrieve and display specific log entries.

The different versions of the syntax are described in the following sections.

Adding a log entry

The following syntax adds an entry to the specified log file. The text for the entry is taken from standard input to the terminal, ending with **Ctrl-D**.

```
dsjob -log [ -info | -warn ] [ -useid ] project job|job_id
```

-info specifies an information message. This is the default if no log entry type is specified.

-warn specifies a warning message.

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the name of the project containing *job*.

job is the name of the job that the log entry refers to. To identify a job invocation, use the format *job.invocation_id.*

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

This syntax is equivalent to the **DSLogEvent** function.

Displaying a short log entry

The following syntax displays a summary of entries in a job log file:

```
dsjob -logsum [-type type] [ -max n ] [ -useid ] project job|job_id
```

-type *type* specifies the type of log entry to retrieve. If **-type** *type* is not specified, all the entries are retrieved. *type* can be one of the following options:

This option...
Retrieves this type of log entry...

INFO Information.

WARNING
Warning.

FATAL
Fatal error.

REJECT
Rejected rows from a Transformer stage.

STARTED
All control logs.

RESET
Job reset.

BATCH
Batch control.

ANY All entries of any type. This is the default if *type* is not specified.

-max *n* limits the number of entries retrieved to *n*.

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the project containing *job*.

job is the job whose log entries are to be retrieved. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

Displaying a specific log entry

The following syntax displays the specified entry in a job log file:

```
dsjob -logdetail [-full] [-useid] project job|job_id entry
```

-full specify this if you want the message ID and the invocation ID fields to be displayed.

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the project containing *job*.

job is the job whose log entries are to be retrieved. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

entry is the event number assigned to the entry. The first entry in the file is 0.

This syntax is equivalent to the **DSGetLogEntry** function, or the **DSGetLogEntryFull** function if **-full** is specified.

Identifying the newest entry

The following syntax displays the ID of the newest log entry of the specified type:

```
dsjob -lognewest [ -useid ] project job|job_id type
```

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the project containing *job*.

job is the job whose log entries are to be retrieved. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

type can be one of the following options:

This option...
Retrieves this type of log entry...

INFO Information

WARNING
Warning

FATAL
Fatal error

REJECT
Rejected rows from a Transformer stage

STARTED
Job started

RESET
Job reset

BATCH
Batch

This syntax is equivalent to the **DSGetNewestLogId** function.

Generating a report

The dsjob command can be used to generate an XML format report containing job, stage, and link information.

dsjob -report [**-useid**] *project job*|*jobid [report_type*]

-useid specify this if you intend to use a job alias (jobid) rather than a job name (job) to identify the job.

project is the project containing *job*.

job specifies the job to be reported on by job name. To identify a job invocation, use the format *job.invocation_id*.

job_id is an alias for the job that has been set using the **dsjob -jobid** command.

report_type is one of the following:

- BASIC - Text string containing start/end time, time elapsed and status of job.
- DETAIL - As basic report, but also contains information about individual stages and links within the job.
- XML - Text string containing full XML report.

The generated report is written to stdout.

This syntax is equivalent to the **DSMakeJobReport** function.

Commands for administering projects

There is a single command for administering projects, `dsadmin`. The command has a large range of options.

These options are described in the following topics:

- The logon clause
- Creating a project
- Deleting a project
- Protecting a project
- Enabling/disabling automatic purging of log files
- Enabling/Disabling the display of generated OSH in parallel jobs.
- Enabling/Disabling runtime column propagation in parallel jobs.
- Enabling/Disabling the availability of job administration features in the Director client for a particular project.
- Setting the advanced runtime options for parallel jobs.
- Setting the base directory name for parallel jobs.
- Setting the deployed job template directory for parallel jobs.
- Setting custom deployment options for parallel jobs.
- Creating a new environment variable.
- Deleting an environment variable.
- Setting the value of an environment variable
- Listing projects on a server.
- Listing project properties.
- Listing environment variables.

Securing credentials in `dsadmin` commands

You can encrypt your credentials and store the encrypted values in files to use in **`dsadmin`** commands. To enter credentials in the command line, you can alternatively use credential prompting to hide your password in the command window.

About this task

You can use encrypted credentials in your **`dsadmin`** commands so that you can avoid typing clear data on the screen when you run commands in the command line. See the *Encrypt command* documentation in the IBM InfoSphere Information Server Administration Guide for information about this command.

Procedure

1. Encrypt your credentials by running the **`encrypt`** command, and copy and save the encrypted values in a credentials file (`*.txt`). See the topic on the credentials file for restrictions and sample contents for the credentials file.
2. Run the **`dsadmin`** command. You can enter your credentials securely by using one of these methods:
   - **Using the credentials file.** If you are running a **`dsadmin`** command that requires your user credentials, run your command with the **`-authfile`** parameter and specify the full path of the credentials file that you want to use. A sample syntax for using the **`-authfile`** parameter in a **`dsadmin`** command is as follows:

```
dsadmin -authfile c:\cred_file.txt
 -listprojects
```

- **Credentials prompting.** If you want to specify your credential data through the command line, specify only the **-url** or the **-domain** parameter, and the **-server** parameter. You are prompted for the user name and password. (The password is hidden in the command window.) If you include the **-url** or the**-domain** parameter, and the **-server** and **-user** parameters in your command, then you are prompted for the password.

The logon clause

By default, the InfoSphere DataStage CLI connects to the engine on the local system using the user name and password of the user running the command.

For the **dsjob** and **dsadmin** commands, you can specify a different domain, engine, user name, or password using the logon clause, which is equivalent to the API **DSSetServerParams** function. Its syntax is as follows:

```
[-url domainURL |
 -domain domain_name ][ -user username ][ -password password ]
     [ -server enginename ]
```

or, for **dsjob** command only:

```
-domain NONE -user username -password password -server enginename
```

domainURL specifies a full format URL for the domain to log on to. The URL includes the protocol, host, and port information for the domain in this format: `https://domain:port`. The port defaults to 9443 if it is not specified.

domain_name specifies the domain to log on to. For **dsjob**, you can set `-domain NONE` to log on to the engine rather than the domain. In this case the user name and password are for the engine, not for the domain.

enginename specifies a different engine to log on to.

username specifies a different user name to use when logging on.

password specifies a different password to use when logging on.

Encrypted user names and passwords are not supported on the command line, use a credentials file if you want to use encrypted values.

If you do not want to type your credentials in the command line and you do not want to use a credentials file, specify only the **-url** or the **-domain** parameter, and the **-server** parameter, and you are prompted for the user name and password. (The password is hidden as you type in the command window.) If you include the **-url** or the **-domain** parameter, and the **-server** and **-user** parameters in your command, then you are prompted for the password. Here is a sample command and interaction:

```
C:\IBM\InformationServer\Clients\Classic>dsjob
-domain [2002:920:c000:217:9:32:217:32]:9443 -server RemoteServer
-ljobs newTest
Please type user name:admin
Please type password:
Job_ODBC

Status code = 0
```

For computers that do not use the default ports, to connect to a project that is on a local server, specify the **-server** option with only the port number and do not specify the server name. You do not need to specify your user name and password. For example:

```
dsjob -server :31539 -lprojects
```

For a more secure login for the **dsjob** and **dsadmin** commands, you can use a credentials file that can contain encrypted data:

```
-authfile credentials_filename
```

credentials_filename is the full path and name of the file that contains the logon details. This file supports encrypted and unencrypted data. See The credentials file for details and sample contents for the file.

You could alternatively specify the unencrypted details in a credentials file by using the following syntax:

```
-file credentials_filename domainname enginename
```

Note: The **-file** credentials file cannot contain encrypted data.

For the **dsjob** command, you can also use the command:

```
-file credentials_filename NONE enginename
```

domainname specifies the domain for which the file contains logon details. For **dsjob**, you can set NONE to log on to the engine rather than the domain. In this case the username and password are for the engine, not for the domain.

enginename specifies the engine for which the file contains logon details.

credentials_filename is the full path and name of the file that contains the logon details. This file supports only unencrypted data. The file should contain the following information if logging on to the domain:

domainname,*enginename*, *username*, *password*

The file must contain the following information if logging on to the engine:

enginename, *username*, *password*

Including the logon clause in your commands can expose your username and password. It is better to use the **-authfile** option and hold the encrypted logon information in a separate file, or to let the computer prompt you for your password.

Creating a project

The *dsadmin* command can be used for creating projects.

You need to have InfoSphere DataStage administrator status in order to use this command:

```
dsadmin -createproject ProjectName
   [-location ProjectLocation]
   [-copyroles SourceProject]
```

ProjectName is the name of the project.

-location *ProjectLocation* is the location of the project in the form of a path name and with the project name included. In the following example, *test* is the project name.

```
dsadmin -createproject test [-location /u1/IS85/IBM/InformationServer/Projects/test]
```

If no location is specified, the project is created in the Projects directory in the server install directory.

-copyroles *SourceProject* is the name of the project from which the user roles for the new project are copied.

Deleting a project

The *dsadmin* command can be used for deleting existing projects.

You need to have InfoSphere DataStage administrator status in order to use this command:

```
dsadmin -deleteproject ProjectName
```

ProjectName is the project to be deleted.

Protecting a project

The **dsadmin** command can be used to protect or unprotect a project.

A protected project is a special category of project and, normally, nothing can be added, deleted, or changed in the project.

Users can view objects in the project, and perform tasks that affect the way that a job runs rather than the design of the job:

- Run jobs
- Set job properties
- Set job parameter default values

Users with Production Manager and Administrator status can import existing InfoSphere DataStage components into a protected project.

You need to have InfoSphere DataStage administrator status to use this command.

To protect a project, run this command:

```
dsadmin -protectproject TRUE ProjectName
```

where *ProjectName* is the name of the project.

To unprotect a project, run this command:

```
dsadmin -protectproject FALSE ProjectName
```

where *ProjectName* is the name of the project.

Enabling/disabling automatic purging of log files

The **dsadmin** command can be used to enable or disable automatic purging of job log files for a project.

Enabling automatic purging of log files

Run this command to enable automatic purging of job log files that are more than a specified number of days old for a project:

```
dsadmin -autopurgelog TRUE -days N ProjectName
```

where

- *N* is the number of days after which job logs are purged. For example, if you specify five days, job log files for job runs that are over five days old are purged.
- *ProjectName* is the project for which automatic purging of job log files is to be enabled.

Run this command to enable automatic purging of job log files when job log files for more than a specified number of job runs exist for a project:

```
dsadmin -autopurgelog TRUE -runs N ProjectName
```

where

- *N* is the number of job runs for which log files that are retained. For example, if you specify 10 job runs, job log files for the last 10 job runs are retained and all older job log files are purged.
- *ProjectName* is the project for which automatic purging of job log files is to be enabled.

Disabling automatic purging of log files

Run this command to disable automatic purging of job log files for a project:

```
dsadmin -autopurgelog FALSE ProjectName
```

where *ProjectName* is the project for which automatic purging of job log files is to be disabled.

Enabling/disabling OSH display

The *dsadmin* command can be used for enabling or disabling the display of generated OSH in parallel jobs.

You need to have InfoSphere DataStage administrator status in order to use this command:

```
dsadmin -oshvisible TRUE | FALSE ProjectName
```

Note: Although this command requires a project name, this setting applies to ALL projects on the server.

This command is only available for parallel jobs.

Enabling/disabling runtime column propagation

The *dsadmin* command can be used for enabling or disabling runtime column propagation in parallel jobs in a particular project.

You need to have InfoSphere DataStage administrator status in order to use this command:

```
dsadmin -enablercp TRUE | FALSE ProjectName
```

ProjectName is the project whose parallel jobs are to have runtime column propagation enabled or disabled.

This command is only available for parallel jobs.

Enabling/disabling job administration from the Director client

The *dsadmin* command can be used for enabling or disabling the job administration features in the Director client for jobs in a particular project.

You need to have InfoSphere DataStage administrator status to use this command:

dsadmin -enablejobadmin TRUE | FALSE *ProjectName*

ProjectName is the project for which job administration in the Director client will be enabled or disabled.

Enabling/disabling generation of XML report

This option is only relevant for parallel jobs being compiled into a deployment package.

The deployment package can include a job report in XML format, and this command enables or disables the generation of this report.

dsadmin -enablegeneratexml TRUE | FALSE *ProjectName*

ProjectName is the project whose parallel jobs are to have XML reports enabled or disabled.

This command is only available for parallel jobs.

Enabling/disabling advanced runtime properties

The *dsadmin* command can be used for setting advanced runtime properties for parallel jobs in a particular project.

You need to have InfoSphere DataStage administrator status in order to use this command:

dsadmin -advancedruntime "*AdvancedRuntimeOptions*" *ProjectName*

ProjectName is the project whose parallel jobs will have the specified advanced runtime options set.

AdvancedRuntimeOptions is the value to set the property to and must be quoted.

This command is only available for parallel jobs.

To unset the properties repeat the command with an empty string, for example:

dsadmin -advancedruntime "" myproject

Setting the base directory

The *dsadmin* command can be used for setting the base directory for parallel jobs in a particular project.

You need to have InfoSphere DataStage administrator status in order to use this command:

dsadmin -basedirectory *BaseDirectoryName ProjectName*

ProjectName is the project whose parallel jobs the base directory is being set for.

BaseDirectoryName is the value to set the property to.

This command is only available for parallel jobs.

Setting the deployment directory template

The *dsadmin* command can be used for setting the deployment directory template for parallel jobs in a particular project.

You need to have InfoSphere DataStage administrator status in order to use this command:

```
dsadmin -deploymentdirectory DirectoryTemplate ProjectName
```

ProjectName is the project whose parallel jobs are having the deployment directory template defined.

DirectoryTemplate is the value to set the property to.

This command is only available for parallel jobs.

Adding an environment variable

The *dsadmin* command can be used for creating a new environment variable in a particular project.

The environment variable is added to the "User Defined" category.

```
dsadmin -envadd EnvVarName -type STRING | ENCRYPTED
-prompt "PromptText" [-value "Value"] ProjectName
```

EnvVarName is the name of the environment variable being created.

-type specified the type of the environment variable and should be set to either STRING or ENCRYPTED.

-prompt *PromptText* is the prompt to be associated with this environment value The *PromptText* must be quoted as it can contain spaces.

-value *Value* is the value for the new environment variable. Value must be quoted. If this is not given, the value for the environment variable will need to be set using the **dsadmin -envset** command.

ProjectName is the project to which the environment variable is being added.

Deleting an environment variable

The *dsadmin* command can be used for deleting an environment variable in a particular project.

It is not possible to delete a built-in environment variables.

```
dsadmin -envdelete EnvVarName ProjectName
```

EnvVarName is the environment variable being deleted.

ProjectName is the project the environment variable is being deleted from.

Setting the value of an environment variable

The *dsadmin* command can be used for setting the value of an environment variable in a particular project.

If setting a list type environment variable (for example, APT_EXECUTION_MODE), then you should set it to one of the permissible internal values, rather than one of the list members as they are shown in the Administrator client. For example, if you wanted to set APT_EXECUTION_MODE so that parallel jobs executed in one process mode, you would set the environment variable value to `ONE_PROCESS', not `One process' as offered in the Administrator client.

If you are setting a boolean type environment variable, set the value to 1 for TRUE and 0 for FALSE.

If you are using $ENV to set the value of an environment variable to its current setting in the environment, then you should use single quotation marks to ensure that it picks up the correct value (for example, dsadmin -envset NEW3 -value '$ENV' dstage).

dsadmin -envset *EnvVarName* **-value** "*Value*" *ProjectName*

EnvVarName is the environment variable whose value is being set.

-value "*Value*" is the value for the environment variable and must be quoted.

ProjectName is the project for which the environment variable is being set.

Listing projects

The *dsadmin* command can be used for listing the projects on an engine tier.

dsadmin *-listprojects*

Lists all the projects on the engine tier.

Listing properties

The *dsadmin* command can be used for listing the properties of a project.

The following properties are listed:

- Whether generated OSH is visible in parallel jobs.
- Whether runtime column propagation is enabled in parallel jobs.
- The base directory name for parallel jobs.
- Advanced runtime options for parallel jobs.
- Custom deployment commands for parallel jobs.
- Deployed job directory template.
- Whether job administration is enabled in the Director client or not.

The parallel job properties will only be listed if parallel jobs are available.

dsadmin -listproperties *ProjectName*

ProjectName is the project for which the properties are to be listed.

Listing environment variables

The *dsadmin* command can be used for listing the environment variables in a project.

dsadmin -listenv *ProjectName*

Commands for importing from .dsx files

You can import objects from .dsx files into the specified repository.

The DSXImportService command has several options. You can use the command to import the contents of an entire .dsx file, or specified objects within a .dsx file, and you can generate a report of the import process. You can also use the DSXImportService command to list the contents of a .dsx file.

You can run the DSXImportService command on any computer that has ASBNode installed.

Import objects from a .dsx file

To run the DSXImportService command, you must specify connection details for the services tier and the path of the file that you want to import.

Purpose

The DSXImportService command imports the objects from a `.dsx` file into an IBM InfoSphere DataStage repository.

Parameters

You can specify domain, user name, and password details for the services tier in the following ways:

- as values in a credentials file, which can be encrypted. Specifying encrypted connection details in a credentials file is the most secure option.
- as unencrypted values in a file. If you specify details in a file, you can also specify the details on the command line to override some of the contents of the file. For example, you can specify a different domain but take the user name and password as specified in the file, so you can use the same file to connect to different computers. Specifying user name and password in a file rather than on the command line gives greater security.
- as unencrypted values on the command line. If you specify only the domain details, you are prompted for the user name and password. (The password is hidden as you type in the command window.) If you specify the domain and user name details, you are prompted for the password.

The following syntax specifies a credentials file containing connection details:

```
-ISAuthFile isAuthFile
-DSProject dsProject
-DSXFile dsxFile
[-Overwrite | -OverwriteReadOnly]
[-Verbose]
[-StopOnError]
[selected_import]
```

The following syntax specifies a file that contains connection details:

```
-ISFile isFile
[-ISHost isHost[:port]]
[-ISUser isUser]
[-ISPassword isPassword]
[–DSHost dsHost[:port]]
-DSProject dsProject
-DSXFile dsxFile
[-Overwrite | -OverwriteReadOnly]
[-Verbose]
[-StopOnError]
[selected_import]
```

The following syntax specifies connection details directly on the command line:

```
-ISHost isHost[:port]
[-ISUser isUser [-ISPassword isPassword]]
[-DSHost dsHost[:port]]
-DSProject dsProject
–DSXFile dsxFile
[-Overwrite | -OverwriteReadOnly]
[-Verbose]
[-StopOnError]
[selected_import]
```

-ISAuthFile *isAuthFile*
Specifies a credentials file that contains connection details for the services tier. The connection details can be encrypted. The credentials file must contain these four arguments:

```
domain=isHost[:port]
user=isUser
password=isPassword
server=dsHost[:port]]
```

The file must contain only these four arguments, with one argument specified per line. The arguments are case-sensitive and take the same format as the command-line arguments.

For more information on credential files including an example, see .

-ISFile *isFile*
Specifies a file name that contains unencrypted connection details for the services tier. If any connection details are specified on the command line, they override those details that are defined within the file. The file specifies the connection details in the following arguments:

```
-ISHost isHost[:port]
-ISUser isUser
-ISPassword isPassword
```

The file must contain only these three arguments, with one argument specified per line. The arguments are case-sensitive and take the same format as the command-line arguments.

-ISHost *isHost***[:***port***]**
Specifies the name of the computer that hosts the domain. If you do not specify a port number, the default port number, 9443, is used.

-ISUser *isUser*
Specifies the name of a user on the domain.

-ISPassword *isPassword*
Specifies the password for the domain user.

-DSHost *dsHost***[:***port***]**
If the engine tier is not installed on the same computer as the services tier, you

must specify the computer that hosts the engine tier. If you do not specify a port number, the default port number is used.

-DSProject *dsProject*
Specifies the project into which the objects are imported.

-DSXFile *dsxFile*
Specifies the `.dsx` file from which to import objects. You can specify a full path name, which can be local or remote.

-Overwrite
Specify this option to overwrite existing objects in the repository. If you do not specify this option, attempting to reimport existing objects causes an error.

-OverwriteReadOnly
This option does the same as **-Overwrite** but additionally replaces any read-only items found. If this option is not specified, existing read-only items are not overwritten.

-Verbose
Specify this option to generate a full report of the objects imported. By default, only import errors are reported.

-StopOnError
Specify this option to stop the import if an error is encountered when you are importing an object.

selected_import
You can specify options here to import selected objects from a `.dsx` file. You specify the object type and the object name as specified in the following table. You can specify a full name or an abbreviated name for the object type.

Table 1. Selected import options

Abbreviated option	Full option	Object type
-JB	-JOB	job
-EJ	-EXECUTABLEJOB	job executable
-DE	-DATAELEMENT	data element
-TD	-TABLEDEFINITION	table definition
-ST	-STAGETYPE	stage type
-TR	-TRANSFORM	transform
-RT	-ROUTINE	routine
-ID	-IMSDATABASE	IMS database
-IV	-IMSVIEWSET	IMS viewset
-MP	-MACHINEPROFILE	machine profiles
-SC	-SHAREDCONTAINER	shared container
-QR	-QSRULEASSEMBLY	QualityStage rule set
-PS	-PARAMETERSET	parameter set
-DC	-DATACONNECTION	data connection

Sample

The following command imports the objects in the `oldproject.dsx` file into the dstage project. Connection details are specified in the `myconnection` file.

```
DSXImportService -ISFile myconnection –DSProject dstage
–DSXFile c:\export\oldproject.dsx
```

The following command imports the objects in the file `newproject.dsx` into the dstage project. Connection details are specified in the command.

```
DSXImportService -ISHost isserver:9443 -ISUser wgamsworth
-ISPassword paddock –DSProject dstage –DSXFile c:\export\newproject.dsx
```

The following command imports selected objects from the `partproject.dsx` file into the dstage project. Connection details are specified in the `myconnection` file.

```
DSXImportService -ISFile myconnection –DSProject dstage
-DSXFile c:\export\oldproject.dsx -JB job1 job2
–TD APTSchemas\CFDImport\XYZ_RECORD –ROUTINE routine1 routine2
```

Listing the contents of a .dsx file

You can use the DSXImportService command to list the contents of a .dsx file.

Purpose

Use the List parameter with the DSXImportService command to list the objects that a .dsx file contains. You do not have to connect to a domain to list the file.

Parameters

The following syntax specifies a file to list:

```
-List -DSXFile <dsxfile>
]
```

–DSXFile *dsxFile*
: Specifies the .dsx file to list. You can specify a full path name, and the path can be local or remote.

Sample

The following command lists the contents of the file oldproject.dsx..

```
DSXImportService -List –DSXFile c:\archives\oldproject.dsx
```

Commands for checking and repairing projects

You can check whether the design-time assets of a project are synchronized with its corresponding InfoSphere DataStage server project assets, and make repairs if necessary.

If the design-time assets for a project that are held in the metadata repository are out of step with the server project, then the server project, or assets within the server project, can become unusable.

You can use the **SyncProject** command to check for inconsistencies, and to repair inconsistencies if any are detected. You might use the command as part of your normal maintenance schedule to check for problems. If any problems are detected, use the command in repair mode to fix the problems. Alternatively, you can use the command interactively and be prompted if repair action is needed.

The **SyncProject** command can detect and attempt to repair, the following issues:

Project missing in metadata repository
: This fault arises if a server project exists, but the corresponding project

does not exist in the metadata repository. **SyncProject** attempts to repair the fault by removing the server project.

Server project missing
This fault arises if a project exists in the metadata repository, but the corresponding server project does not exist. **SyncProject** attempts to repair the fault by recreating the server project. If **SyncProject** succeeds in recreating the project, any jobs that the project contains must be recompiled.

Server project files are corrupted
The server project holds descriptions of project objects in files. If these files are corrupted, **SyncProject** attempts to resolve the issue by returning the corrupted project file to its initial state.

Job or shared container missing from the metadata repository
This fault arises if a server job or shared container exists, but the corresponding metadata repository job or shared container does not exist. **SyncProject** attempts to repair the fault by removing the server job or shared container.

Server job or shared container missing
This fault arises if a job or shared container exists in the metadata repository, but the corresponding server job or shared container does not exist. **SyncProject** attempts to repair the fault by recreating the server job or shared container. If **SyncProject** succeeds in recreating the job, it must be recompiled.

Corrupted server engine job files
This fault arises if the files in which the server engine defines an executable job and its state become corrupted. To recover a corrupted file, **SyncProject** recreates an empty file. As a consequence, the job might need to be recompiled and the job log history and current job status information might be lost.

Incorrect server job or shared container folder
This fault arises if a job or shared container is in a different folder on the server compared to the metadata repository. **SyncProject** attempts to repair the fault by updating the folder attribute of the job or shared container in the server project.

If the **SyncProject** command cannot repair a server project, you can use the **SyncProject** command to reconstruct the project from the metadata repository.

Authentication parameters for SyncProject command

When you use the **SyncProject** command to check or to fix your projects, you must specify authentication details.

Parameters

You can specify services tier host, user name, and password details in a file, or you can specify the details directly on the command line. If you specify details in a file, you can also specify the details on the command line to override some of the contents of the file. For example, you can specify a different host but take the user name and password as specified in the file, so you can use the same file to connect to different computers. Specifying user name and password in a file rather than on the command line gives greater security.

The following syntax specifies a file that contains connection details:

```
SyncProject -ISFile isFile
[-ISHost isHost[:port]]
[-ISUser isUser]
[-ISPassword isPassword]
[–DSHost dsHost[:port]]
```

The following syntax specifies connection details directly on the command line:

```
SyncProject
-ISHost isHost[:port]
-ISUser isUser
-ISPassword isPassword
[–DSHost dsHost[:port]]
```

-ISFile *isFile*
Specifies the file name that contains the connection details. This parameter provides a level of security by hiding the login details from view. If you use this parameter, you do not have to provide the connection details on the command line. If any connection details are specified on the command line, however, they override those connection details defined within the file. The file specifies the connection details in the following arguments:

```
-ISHost isHost -ISUser isUser -ISPassword isPassword
```

The content of the file requires one argument per line. The following arguments are case-sensitive.

-ISHost *isHost***[:***port***]**
Specifies the name of the computer that hosts the domain. If you do not specify a port number, the default port number, 9443, is used.

-ISUser *isUser*
Specifies the name of a user on the services tier.

-ISPassword *isPassword*
Specifies the password for the services tier user.

-DSHost *dsHost***[:***port***]**
If the engine tier is not installed on the same computer as the services tier, you must specify the computer that hosts the engine. If you do not specify a port number, the default port number is used.

Checking projects for inconsistencies

Use the **SyncProject** command to check that your projects are consistent between design-time and run time repositories.

Parameters

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname...
[-JobJobname...]
-Report [filename]
[-StopOnError]
```

-Project *Projectname...*
Specifies one or more projects to be checked.

-Job *Jobname...*
If you specify only one project in your command, specifies a list of jobs within that project to check.

-report [*filename***]**
Specifies that a report is required. You can optionally specify a file name, and the report is written to this file.

Examples

The following command requests a consistency report for the project named dstage3. In this case both the services tier and the engine tier are on the computer named R101.

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword -project dstage3
-report
```

The command outputs the following report:

```
DSEngine Restorer Report

Feb 05, 2010 9:32:00 AM

IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158

DataStage Project: dstage3
--------------------------

0 Issues Found.

Overall Summary
---------------
0 Issues found.
```

The following command requests a consistency report for all the projects with the name dstage*n*. The command returns results for the projects dstage3, dstage4, dstage5, and dstage9. The report is written to a file as well as being output to the screen.

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword -project dstage*
-report c:\myprojrep
```

In this case, two inconsistencies are found in the project named dstage9. The following report is output, and written to the file c:\myprojrep:

```
DSEngine Restorer Report

Feb 05, 2010 9:39:00 AM

IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158

DataStage Project: dstage3
--------------------------

0 Issues Found.

DataStage Project: dstage4
--------------------------

0 Issues Found.
```

```
DataStage Project: dstage5
--------------------------

0 Issues Found.

DataStage Project = dstage9
---------------------------

2 Issues Found.

DS Engine Job 'testJob' is missing.
DS Engine Job 'testJob2' category 'incorrectCategory' should be 'correctCategory'

Overall Summary
---------------
2 Issues found.
```

Repairing inconsistent projects

You can use the **SyncProject** command with the -fix parameter to repair projects that have reported inconsistencies.

Parameters

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname...
[-Job Jobname...]
-Fix [filename]
-StopOnError
```

-Project *Projectname...*
: Specify one or more projects to be repaired.

-Job *Jobname...*
: If you specify only one project in your command, you can specify a list of jobs within that project to repair.

-Fix [*filename***]**
: Specifies that fixes are required for faults previously reported in the named project or projects. You can optionally specify a file name, and the results of any attempted fixes are written to the file.

-StopOnError
: Specify this parameter to halt if **SyncProject** fails to fix an inconsistency. No more fixes are attempted.

Examples

The following command requests that fixes are made to the projects dstage3 and dstage5.

```
SyncProject -ISFile islogin -project dstage3 dstage5 -Fix
```

The command is able to make the necessary repairs outputs the following report.

```
DSEngine Restorer Fix Results

Feb 05, 2009 9:39:00 AM

IS Host = R101
IS Port = 9443
IS User = admin
```

```
DS Host = R101
DS Port = 3158

DataStage Project: dstage3
--------------------------

RESOLVED: DS Engine Job 'testJob' is missing.
RESOLVED: DS Engine Job 'testJob2' category 'incorrectCategory'
should be 'correctCategory'.

2 Issues resolved.
0 Issues remaining.

DataStage Project: dstage5
--------------------------

RESOLVED: DS Engine Job 'test2Job' is missing.

1 Issues resolved.
0 Issues remaining.

Overall Summary
---------------
3 Issues resolved.
0 Issues remaining.
```

The following command requests that fixes are made to the project dstage6, and the results written the file c:\fixresults:

```
SyncProject -ISFile islogin -project dstage6 -fix c:\fixresults
```

The command is not able to make the necessary repairs. The command outputs the following report and writes it to the file c:\fixresults:

```
DSEngine Restorer Fix Results

Feb 05, 2009 9:39:00 AM

IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158

DataStage Project: dstage6
--------------------------

UNRESOLVED: DS Engine Job 'nexttestJob' is missing.
UNRESOLVED: DS Engine Job 'nexttestJob2' folder 'incorrectFolder'
should be 'correctFolder'.

Overall Summary
---------------
0 Issues resolved.
2 Issues remaining.
```

Interactive checking and repairing projects

Use the **SyncProject** command interactively to check that run time assets of a project match its design-time assets. If an inconsistency is found, you are prompted to decide whether to attempt to fix it.

Parameters

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname...
[-JobJobname...]
-Report [filename]
-Fix [filename]
-StopOnError
```

-Project *Projectname...*
Specify one or more projects to be checked.

-Job*Jobname...*
If you specify only one project in your command, you can specify a list of jobs within that project to check.

-report [*filename***]**
Specifies that a report is required. You can optionally specify a file name, and the report is written to this file.

-fix [*filename***]**
Specifies that any inconsistencies are fixed. You can optionally specify a file name, and information about the fix, and its success or failure, is written to this file.

-StopOnError
Specify this option to halt if SyncProject fails to fix an inconsistency.

Examples

The following command requests a consistency report for the project named dstage3 and that any inconsistencies found are repaired:

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword
    -Project dstage3 -Report -Fix
```

The following report is output:

```
DSEngine Restorer Report

Feb 05, 2009 9:39:00 AM

IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158

DataStage Project: dstage3
--------------------------

2 Issues Found.

DS Engine Job 'testJob' is missing.
DS Engine Job 'testJob2' category 'incorrectCategory'
should be 'correctCategory'

Overall Summary
---------------
2 Issues found.
```

You are prompted whether to fix the detected issues. SyncProject then reports on the success of the fix as follows:

```
DSEngine Restorer Fix Results

Feb 05, 2009 9:39:00 AM
```

```
IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158

DataStage Project: dstage3
--------------------------

RESOLVED: DS Engine Job 'testJob' is missing.
RESOLVED: DS Engine Job 'testJob2' category 'incorrectCategory'
should be 'correctCategory'.

2 Issues resolved.
0 Issues remaining.

Overall Summary
---------------
2 Issues resolved.
0 Issues remaining.
```

Reconstructing a project

Use the **SyncProject** command to reconstruct a project from the repository.

Parameters

The project directory is deleted before the project is reconstructed from the repository. The reconstructed project does not retain any log or job status data.

The jobs and routines in the reconstructed project must be recompiled.

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname
-Reconstruct
```

-Project *Projectname*
: Specifies the project to be reconstructed.

-Reconstruct
: Specifies that a project is to be reconstructed.

Example

The following command requests that a project named dstage1 is reconstructed from the repository.

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword -project dstage1
-reconstruct
```

The command outputs the following report:

```
DSEngine Restorer Fix Results

Wed Jan 20 09:13:35 GMT 2010

IS Host = R101
IS Port = 9443
IS User = admin

DS Host = R101
DS Port = 3158
```

```
DataStage Project: dstage1
--------------------------

RESOLVED: Deleting and re-creating DS Engine Project: dstage1.
RESOLVED: DS Engine Job 'px1' is missing.
RESOLVED: DS Engine Job 'sv1' is missing.
RESOLVED: DS Engine Shared Container 'pxsc1' is missing.
RESOLVED: DS Engine Shared Container 'svsc1' is missing.

Overall Summary
---------------

5 Issues Resolved.
0 Issues Unresolved.
```

Backing up a project

Use the **SyncProject** command to make a backup copy of a project.

Parameters

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname
-Backup [filename]
```

-Project *Projectname*
: Specifies the project to be backed up.

-Backup [*filename***]**
: Optionally, specify the full path name for the tar archive file that is created. If you do not specify a file name, the tar archive file is called *Projectname*.tar

Example

The following command requests that a backup copy is made of the project named dstage3 to a tar archive file called dstage3.tar.

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword -project dstage3
-backup c:\archives\dstage3.tar
```

The command outputs the following report:

```
Performing the backup.
The backup was successful.
```

Restoring a project

Use the **SyncProject** command to restore a project from a backup copy.

Parameters

The **SyncProject** command has the following syntax:

```
SyncProject authentication_parameters
-Project Projectname
-Restore filename
```

-Project *Projectname*
: Specifies the project to be restored.

-Restore [*filename***]**
: Specifies the name of the tar archive file that you previously created.

Example

The following command requests that a project named dstage3 is restored from a tar archive file called `dstage3.tar`.

```
SyncProject -ISHost R101:9443 -ISUser admin -ISPassword pword -project dstage3
-restore dstage3.tar
```

The command outputs the following report:

```
Performing the project restore.
The project restore was successful.
```

Chapter 2. InfoSphere DataStage Development Kit (Job Control Interfaces)

The InfoSphere DataStage development kit and other job control interfaces are tools that you can use to run jobs directly on the engine. These tools provide flexibility for running jobs in multiple scenarios without the need to use the Director client.

InfoSphere DataStage provides a range of methods that enable you to run server or parallel jobs directly on the engine, without using the Director client. The methods are:

- C/C++ API (the InfoSphere DataStage development kit)
- InfoSphere DataStage BASIC calls
- Command line Interface commands (CLI)
- InfoSphere DataStage macros

These methods can be used in different situations as follows:

- API. Using the API you can build a self-contained program that can run anywhere on your system, provided that it can connect to an engine tier host across the network.
- BASIC. Programs built using the InfoSphere DataStage BASIC interface can be run from any engine tier host on the network. You can use this interface to define jobs that run and control other jobs. The controlling job can be run from the Director client like any other job, or directly on the engine machine from the TCL prompt.
- CLI. The CLI can be used from the command line of any engine tier host on the network. Using this method, you can run jobs on other engines too.
- Macros. A set of macros can be used in job designs or in BASIC programs. These are mostly used to retrieve information about other jobs.

InfoSphere DataStage Development Kit

Use the InfoSphere DataStage Development Kit for access to the InfoSphere DataStage API, a C or C++ application programming interface

.

This section gives general information about using the InfoSphere DataStage API.

The dsapi.h Header File

Include the InfoSphere DataStage API header file with all API programs.

DataStage API provides a header file that should be included with all API programs. The header file includes prototypes for all InfoSphere DataStage API functions. It is located in the directory *$DSHOME/include* Their format depends on which tokens you have defined:

- If the **_STDC_** or **WIN32** tokens are defined, the prototypes are in ANSI C style.
- If the **_cplusplus** token is defined, the prototypes are in C++ format with the declarations surrounded by:

```
extern "C" {...}
```

- Otherwise the prototypes are in Kernighan and Ritchie format.

Data Structures, Result Data, and Threads

InfoSphere DataStage API functions return information about objects as pointers to data items. This is either done directly, or indirectly by setting pointers in the elements of a data structure that is provided by the caller.

Each thread within a calling application is allocated a separate storage area. Each call to an InfoSphere DataStage API routine overwrites any existing contents of this data area with the results of the call, and returns a pointer into the area for the requested data.

For example, the **DSGetProjectList** function obtains a list of InfoSphere DataStage projects, and the **DSGetProjectInfo** function obtains a list of jobs within a project. When the **DSGetProjectList** function is called it retrieves the list of projects, stores it in the thread's data area, and returns a pointer to this area. If the same thread then calls **DSGetProjectInfo**, the job list is retrieved and stored in the thread's data area, overwriting the project list. The job list pointer in the supplied data structure references the thread data area.

This means that if the results of an InfoSphere DataStage API function need to be reused later, the application should make its own copy of the data before making a new InfoSphere DataStage API call. Alternatively, the calls can be used in multiple threads.

InfoSphere DataStage API stores errors for each thread: a call to the **DSGetLastError** function returns the last error generated within the calling thread.

Writing InfoSphere DataStage API Programs

You write InfoSphere DataStage API programs by following a simple, logical process that uses functions effectively and in the right sequence.

About this task

Your application should use the InfoSphere DataStage API functions in a logical order to ensure that connections are opened and closed correctly, and jobs are run effectively. The following procedure suggests an outline for the program logic to follow, and which functions to use at each step:

Procedure

1. If required, set the host name, user name, and password to use for connecting to InfoSphere DataStage (**DSSetServerParams**).
2. Obtain the list of valid projects (**DSGetProjectList**).
3. Open a project (**DSOpenProject**).
4. Obtain a list of jobs (**DSGetProjectInfo**).
5. Open one or more jobs (**DSOpenJob**).
6. List the job parameters (**DSGetParamInfo**).
7. Lock the job (**DSLockJob**).
8. Set the job's parameters and limits (**DSSetJobLimit**, **DSSetParam**).
9. Start the job running (**DSRunJob**).

10. Poll for the job or wait for job completion (**DSWaitForJob**, **DSStopJob**, **DSGetJobInfo**).
11. Unlock the job (**DSUnlockJob**).
12. Display a summary of the job's log entries (**DSFindFirstLogEntry**, **DSFindNextLogEntry**).
13. Display details of specific log events (**DSGetNewestLogId**, **DSGetLogEntry**, **DSGetLogEntryFull**).
14. Examine and display details of job stages (**DSGetJobInfo** - stage list, **DSGetStageInfo**).
15. Examine and display details of links within active stages (**DSGetStageInfo** - link list, **DSGetLinkInfo**).
16. Close all open jobs (**DSCloseJob**).
17. Detach from the project (**DSCloseProject**).

Building an InfoSphere DataStage API Application

Create an application that uses the InfoSphere DataStage API by writing the program, compiling the code, and linking the application.

About this task

Everything you need to create an application that uses the InfoSphere DataStage API is in a subdirectory called `dsdk` (DataStage Development Kit) in the `IBM\InformationServer\Server` installation directory on the engine tier host machine.

Procedure

1. Write the program, including the `dsapi.h` header file in all source modules that uses the InfoSphere DataStage API.
2. Compile the code. Ensure that the **WIN32** token is defined. (This happens automatically in the Microsoft Visual C/C++ compiler environment.)
3. Link the application, including `vmdsapi.lib`, in the list of libraries to be included.

Redistributing Applications

Redistribute applications, DLLs, or libraries when you run InfoSphere DataStage API applications, depending on the role of the computer the application is installed on.

If you intend to run your InfoSphere DataStage API application on a computer where the engine tier is installed, you do not need to include InfoSphere DataStage API DLLs or libraries as these are installed as part of the engine tier.

If you want to run the application from a computer used as an InfoSphere DataStage client, you should redistribute the following library with your application:

```
vmdsapi.dll
```

If you intend to run the program from a computer that has neither the engine tier nor any InfoSphere DataStage client installed, in addition to the library mentioned above, you should also redistribute the following:

```
dsclnt32.dll dsrpc32.dll ACS_common_cpp.dll ACS_client_cpp.dll
invocation_cpp.dll xmogrt.dll
```

You should locate these files where they will be in the search path of any user who uses the application, for example, in the `%SystemRoot%\System32` directory.

API Functions

Use the functions provided in the InfoSphere DataStage API to perform various tasks with projects and jobs.

This section details the functions provided in the InfoSphere DataStage API. These functions are described in alphabetical order. The following table briefly describes the functions categorized by usage:

Usage	Function	Description
Accessing projects	**DSCloseProject**	Closes a project that was opened with **DSOpenProject**.
	DSGetProjectList	Retrieves a list of all projects on the engine
	DSGetProjectInfo	Retrieves a list of jobs in a project.
	DSOpenProject	Opens a project.
	DSSetServerParams	Sets the host name, user name, and password to use for a job.
Accessing jobs	**DSCloseJob**	Closes a job that was opened with **DSOpenJob**.
	DSGetJobInfo	Retrieves information about a job, such as the date and time of the last run, parameter names, and so on.
	DSLockJob	Locks a job prior to setting job parameters or starting a job run.
	DSOpenJob	Opens a job.
	DSRunJob	Runs a job.
	DSStopJob	Aborts a running job.
	DSUnlockJob	Unlocks a job, enabling other processes to use it.
	DSWaitForJob	Waits until a job has completed.
Accessing job parameters	**DSGetParamInfo**	Retrieves information about a job parameter.
	DSSetJobLimit	Sets row processing and warning limits for a job.
	DSSetParam	Sets job parameter values.
Accessing stages	**DSGetStageInfo**	Retrieves information about a stage within a job.

Usage	Function	Description
Accessing links	**DSGetLinkInfo**	Retrieves information about a link of an active stage within a job.
Accessing log entries	**DSFindFirstLogEntry**	Retrieves entries in a log that meet the specified criteria.
	DSFindNextLogEntry	Finds the next log entry that meets the criteria specified in **DSFindFirstLogEntry**.
	DSGetLogEntry	Retrieves the specified log entry.
	DSGetLogEntryFull	Retrieves the specified log entry, including the message ID and the invocation ID.
	DSGetLogEventIds	Retrieves a list of event log IDs for a given job invocation.
	DSGetNewestLogId	Retrieves the newest entry in the log.
	DSLogEvent	Adds a new entry to the log.
Administering Projects and jobs	DSAddEnvVar	Adds a new environment variable.
	DSAddProject	Add a project.
	DSDeleteEnvVar	Delete an environment variable.
	DSDeleteProject	Delete a project.
	DSListEnvVars	List environment variables.
	DSListProjectProperties	List the properties of a project.
	DSSetEnvVar	Set an environment variable.
	DSSetProjectProperty	Set property for a project.
Handling errors	**DSGetLastError**	Retrieves the last error code value generated by the calling thread.
	DSGetLastErrorMsg	Retrieves the text of the last reported error.

DSAddEnvVar

Use the DSAddEnvVar function to add an environment variable to the specified project. The variable is added to the User Defined category.

Syntax

```
int DSAddEnvVar(   DSPROJECT hProject,
      char *EnvVarName,
      char *Type,
      char *PromptText,
      char *Value);
```

Parameters

hProject is the value returned from **DSOpenProject**

EnvVarName is the name of the environment variable

Type is DSA_ENVVAR_TYPE_STRING for string type environment variables or DSA_ENVVAR_TYPE_ENCRYPTED for encrypted environment variables.

PromptText is the user-visible text describing the environment variable.

Value is value to set the environment variable to or "".

Return Values

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_BADENVVARNAME invalid environment variable name
- DSJE_BADENVVARTYPE invalid type
- DSJE_BADENVVARPROMPT no prompt supplied
- DSJE_READENVVARDEFNS failed to read environment variable definitions
- DSJE_READENVVARVALUES failed to read environment variable values
- DSJE_WRITEENVVARDEFNS failed to write environment variable definitions
- DSJE_WRITEENVVARVALUES failed to write environment variable values
- DSJE_DUPENVVARNAME environment variable already exists
- DSJE_ENCODEFAILED failed to encode an encrypted value

Remarks

To use this method, the program needs to have previously attached to a project using **DSOpenProject**. This returns a handle to the project, *hProject*.

DSAddProject

Use the DSAddProject function to create a new project. The user who runs the code that contains this function must be an InfoSphere DataStage administrator.

Syntax

```
int DSAddProject(
  char *ProjectName,
  char *ProjectLocation);
```

Parameters

ProjectName is the name of the project to create.

ProjectLocation is the path name of the directory to create the project in. To create a project in the default project directory (*install path*/Projects/*projectName*), this argument should be set to "".

Return Values

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_NOTADMINUSER user is not an administrator
- DSJE_ISADMINFAILED failed to determine whether user is an administrator
- DSJE_BADPROJNAME invalid project name supplied
- DSJE_GETDEFAULTPATHFAILED failed to determine default project directory
- DSJE_BADPROJLOCATION invalid path name supplied
- DSJE_INVALIDPROJECTLOCATION invalid path name supplied
- DSJE_OPENFAILED failed to open UV.ACCOUNT file
- DSJE_READUFAILED failed to lock project create lock record
- DSJE_ADDPROJECTBLOCKED another user is adding a project
- DSJE_ADDPROJECTFAILED failed to add project
- DSJE_LICENSEPROJECTFAILED failed to license project
- DSJE_RELEASEFAILED failed to release project create lock record

DSCloseJob

Use the DSCloseJob function to close a job that was opened by using the DSOpenJob function.

Syntax

```
int DSCloseJob(
     DSJOB JobHandle
);
```

Parameter

JobHandle is the value returned from **DSOpenJob**.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is:

DSJE_BADHANDLE Invalid *JobHandle.*

Remarks

If the job is locked when **DSCloseJob** is called, it is unlocked.

If the job is running when **DSCloseJob** is called, the job is allowed to finish, and the function returns a value of DSJE_NOERROR immediately.

DSCloseProject

Use the DSCloseProject function to closes a project that was opened by using the DSOpenProject function.

Syntax

```
int DSCloseProject(
   DSPROJECT ProjectHandle
);
```

Parameter

ProjectHandle is the value returned from **DSOpenProject**.

Return Value

This function always returns a value of DSJE_NOERROR.

Remarks

Any open jobs in the project are closed, running jobs are allowed to finish, and the function returns immediately.

DSDeleteEnvVar

Use the DSDeleteEnvVar function to delete a user-defined environment variable in a specified project.

Syntax

```
int DSDeleteEnvVar(
      DSPROJECT hProject,
      char *EnvVar
);
```

Parameters

hProject is the value returned from **DSOpenProject**

EnvVarName is the name of the environment variable

Return Values

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_READENVVARDEFNS failed to read environment variable definitions
- DSJE_READENVVARVALUES failed to read environment variable values
- DSJE_BADENVVAR environment variable does not exist
- DSJE_WRITEENVVARDEFNS failed to write environment variable definitions
- DSJE_WRITEENVVARVALUES failed to write environment variable values
- DSJE_NOTUSERDEFINED environment variable is not user-defined and therefore cannot be deleted

If the function fails, then the return value is one of the following:

DSDeleteProject

Use the DSDeleteProject function to deletes a project. The user who runs the code that contains this function must be an InfoSphere DataStage administrator. Any jobs that are scheduled to be run that are included in this project are also deleted.

Syntax

```
int DSDeleteProject(
       char *ProjectName
);
```

Parameter

ProjectName is the name of the project to delete.

Return Value

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_NOTADMINUSER user is not an administrator
- DSJE_ISADMINFAILED failed to determine whether user is an administrator
- DSJE_OPENFAILED failed to open UV.ACCOUNT file
- DSJE_READUFAILED failed to lock project record
- DSJE_RELEASEFAILED failed to release project record
- DSJE_LISTSCHEDULEFAILED failed to get list of scheduled jobs for project
- DSJE_CLEARSCHEDULEFAILED failed to clear scheduled jobs for project
- DSJE_DELETEPROJECTBLOCKED project locked by another user
- DSJE_NOTAPROJECT failed to log to project
- DSJE_ACCOUNTPATHFAILED failed to get account path
- DSJE_LOGTOFAILED failed to log to UV account
- DSJE_DELPROJFAILED failed to delete project definition
- DSJE_DELPROJFILESFAILED failed to delete project files

DSFindFirstLogEntry

Use the DSFindFirstLogEntry function to retrieve all the log entries that meet the specified criteria, and write the first entry to a data structure. You can the read subsequent log entries by using the DSFindNextLogEntry function.

Syntax

```
int DSFindFirstLogEntry(
   DSJOB JobHandle,
   int EventType,
   time_t StartTime,
   time_t EndTime,
   int MaxNumber,
   DSLOGEVENT *Event);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

EventType is one of the following keys:

This key...
: **Retrieves this type of message...**

DSJ_LOGINFO
: Information

DSJ_LOGWARNING
Warning

DSJ_LOGFATAL
Fatal

DSJ_LOGREJECT
Transformer row rejection

DSJ_LOGSTARTED
Job started

DSJ_LOGRESET
Job reset

DSJ_LOGBATCH
Batch control

DSJ_LOGOTHER
All other log types

DSJ_LOGANY
Any type of event

StartTime limits the returned log events to those that occurred on or after the specified date and time. Set this value to 0 to return the earliest event.

EndTime limits the returned log events to those that occurred before the specified date and time. Set this value to 0 to return all entries up to the most recent.

MaxNumber specifies the maximum number of log entries to retrieve, starting from the latest.

Event is a pointer to a data structure to use to hold the first retrieved log entry.

Return Values

If the function succeeds, the return value is DSJE_NOERROR, and summary details of the first log entry are written to *Event*.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOMORE
There are no events matching the filter criteria.

DSJE_NO_MEMORY
Failed to allocate memory for results from engine.

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADTYPE
Invalid *EventType* value.

DSJE_BADTIME
Invalid *StartTime* or *EndTime* value.

DSJE_BADVALUE
Invalid *MaxNumber* value.

Remarks

The retrieved log entries are cached for retrieval by subsequent calls to **DSFindNextLogEntry**. Any cached log entries that are not processed by a call to **DSFindNextLogEntry** are discarded at the next **DSFindFirstLogEntry** call (for any job), or when the project is closed.

Note: The log entries are cached by project handle. Multiple threads using the same open project handle must coordinate access to **DSFindFirstLogEntry** and **DSFindNextLogEntry**.

DSFindNextLogEntry

Use the DSFindNextLogEntry function to retrieve the next log entry from the cache.

Syntax

```
int DSFindNextLogEntry(
   DSJOB JobHandle,
   DSLOGEVENT *Event
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

Event is a pointer to a data structure to use to hold the next log entry.

Return Values

If the function succeeds, the return value is DSJE_NOERROR and summary details of the next available log entry are written to *Event*.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOMORE
All events matching the filter criteria have been returned.

DSJE_SERVER_ERROR
Internal error. The engine returned invalid data.

Remarks

This function retrieves the next log entry from the cache of entries produced by a call to **DSFindFirstLogEntry**.

Note: The log entries are cached by project handle. Multiple threads using the same open project handle must coordinate access to **DSFindFirstLogEntry** and **DSFindNextLogEntry**.

DSGetCustInfo

Use the DSGetCustInfo function to obtain information reported at the end of the execution of certain parallel stages. The information collected, and available to be interrogated, is specified at design time. For example, Transformer stage information is specified in the **Triggers** tab in the Transformer stage Properties dialog box.

Syntax

```
int DSGetCustInfo(   DSJOB JobHandle,
      char *StageName,
      char *CustinfoName
      int InfoType,
      DSSTAGEINFO *ReturnInfo
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

StageName is a pointer to a null-terminated string specifying the name of the stage to be interrogated.

Custinfo*Name* is a pointer to a null-terminated string specifying the name of the variable to be interrogated (as set up on the **Triggers** tab).

InfoType is one of the following keys:

This key...
Returns this information...

DSJ_CUSTINFOVALUE
The value of the specified variable.

DSJ_CUSTINFODESC
Description of the variable.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There are no instances of the requested information in the stage.

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADSTAGE
StageName does not refer to a known stage in the job.

DSJE_BADCUSTINFO
CustinfoName does not refer to a known *custinfo* item.

DSJE_BADTYPE
Invalid *InfoType.*

DSGetJobInfo

Use the DSGetJobInfo function to retrieve information about the status of a job.

Syntax

```
int DSGetJobInfo(   DSJOB JobHandle,
      int InfoType,
      DSJOBINFO *ReturnInfo
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

InfoType is a key indicating the information to be returned and can have any of the following values:

This key...
Returns this information...

DSJ_JOBSTATUS
The current status of the job.

DSJ_JOBNAME
The name of the job referenced by *JobHandle*.

DSJ_JOBCONTROLLER
The name of the job controlling the job referenced by *JobHandle*.

DSJ_JOBSTARTTIMESTAMP
The date and time when the job started.

DSJ_JOBWAVENO
The wave number of last or current run.

DSJ_JOBDESC
The Job Description specified in the Job Properties dialog box.

DSJ_JOBFULLDESSC
The Full Description specified in the Job Properties dialog box.

DSJ_JOBDMISERVICE
Set to true if this is a Web service job.

DSJ_JOBMULTIINVOKABLE
Set to true if this job supports multiple invocations.

DSJ_PARAMLIST
A list of job parameter names. Separated by nulls.

DSJ_STAGELIST
A list of active stages in the job. Separated by nulls.

DSJ_USERSTATUS
The value, if any, set as the user status by the job.

DSJ_JOBCONTROL
Whether a stop request has been issued for the job referenced by *JobHandle.*

DSJ_JOBPID
Process id of DSD.RUN process.

DSJ_JOBLASTTIMESTAMP
The date and time when the job last finished.

DSJ_JOBINVOCATIONS
List of job invocation ids. The ids are separated by nulls.

DSJ_JOBINTERIMSTATUS
The status of a job after it has run all stages and controlled jobs, but before it has attempted to run an after-job subroutine. (Designed to be used by an after-job subroutine to get the status of the current job.)

DSJ_JOBINVOCATIONID
Invocation name of the job referenced by *JobHandle*.

DSJ_JOBDESC
A description of the job.

DSJ_STAGELIST2
A list of passive stages in the job. Separated by nulls.

DSJ_JOBELAPSED
The elapsed time of the job in seconds.

ReturnInfo is a pointer to a **DSJOBINFO** data structure where the requested information is stored. The **DSJOBINFO** data structure contains a union with an element for each of the possible return values from the call to **DSGetJobInfo**. For more information, see "Data Structures" on page 75.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There are no instances of the requested information in the job.

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADTYPE
Invalid *InfoType.*

Remarks

For controlled jobs, this function can be used either before or after a call to **DSRunJob**.

DSGetLastError

Use the DSGetLastError function to return the calling thread's last error code value.

Syntax

```
int DSGetLastError(void);
```

Return Values

The return value is the last error code value. The "Return Values" section of each reference page notes the conditions under which the function sets the last error code.

Remarks

Use **DSGetLastError** immediately after any function whose return value on failure might contain useful data, otherwise a later, successful function might reset the value back to 0 (DSJE_NOERROR).

Note: Multiple threads do not overwrite each other's error codes.

DSGetLastErrorMsg

Use the DSGetLastErrorMsg function to retrieve the text of the last reported error from the engine.

Syntax

```
char *DSGetLastErrorMsg(
   DSPROJECT ProjectHandle
);
```

Parameter

ProjectHandle is either the value returned from **DSOpenProject** or NULL.

Return Values

The return value is a pointer to a series of null-terminated strings, one for each line of the error message associated with the last error generated by the engine in response to an InfoSphere DataStage API function call. Use **DSGetLastError** to determine what the error number is.

The following example shows the buffer contents with <null> representing the terminating NULL character:

```
line1<null>line2<null>line3<null><null>
```

The **DSGetLastErrorMsg** function returns NULL if there is no error message.

Remarks

If *ProjectHandle* is NULL, this function retrieves the error message associated with the last call to **DSOpenProject** or **DSGetProjectList**, otherwise it returns the last message associated with the specified project.

The error text is cleared following a call to **DSGetLastErrorMsg**.

Note: The text retrieved by a call to **DSGetLastErrorMsg** relates to the last error generated by the engine and not necessarily the last error reported back to a thread using InfoSphere DataStage API. Multiple threads using InfoSphere DataStage API must cooperate in order to obtain the correct error message text.

DSGetLinkInfo

Use the DSGetLinkInfo function to retrieve information that relates to a specific link of the specified active stage of a job.

Syntax

```
int DSGetLinkInfo(
   DSJOB JobHandle,
   char *StageName,
   char *LinkName,
   int InfoType,
   DSLINKINFO *ReturnInfo);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

StageName is a pointer to a null-terminated character string specifying the name of the active stage to be interrogated.

LinkName is a pointer to a null-terminated character string specifying the name of a link (input or output) attached to the stage.

InfoType is a key indicating the information to be returned and is one of the following values:

Value Description

DSJ_LINKLASTERR
Last error message reported by the link.

DSJ_LINKNAME
Name of the link.

DSJ_LINKROWCOUNT
Number of rows that have passed down the link.

DSJ_LINKEXTROWCOUNT
Number of rows that have passed down the link, returned as a string in the rowCountList field of the DSLINKINFO structure. Use this value when there are more rows returned than supported by the integer field DSJ_LINKROWCOUNT.

DSJ_LINKSQLSTATE
SQLSTATE value from last error message.

DSJ_LINKDBMSCODE
DBMSCODE value from last error message.

DSJ_LINKDESC
Description of the link.

DSJ_LINKSTAGE
Name of the stage at the other end of the link.

DSJ_INSTROWCOUNT
Null-separated list of row counts, one per instance for parallel jobs.

ReturnInfo is a pointer to a **DSJOBINFO** data structure where the requested information is stored. The **DSJOBINFO** data structure contains a union with an element for each of the possible return values from the call to **DSGetLinkInfo**.

Return Value

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There is no instance of the requested information available.

DSJE_BADHANDLE
JobHandle was invalid.

DSJE_BADTYPE
InfoType was unrecognized.

DSJE_BADSTAGE
StageName does not refer to a known stage in the job.

DSJE_BADLINK
LinkName does not refer to a known link for the stage in question.

Remarks

This function can be used either before or after a call to **DSRunJob**.

DSGetLogEntry

Retrieves detailed information about a specific entry in a job log.

Syntax

```
int DSGetLogEntry(
     DSJOB JobHandle,
     int EventId,
     DSLOGDETAIL *Event
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

EventId is the identifier for the event to be retrieved, see "Remarks."

Event is a pointer to a data structure to hold details of the log entry.

Return Values

If the function succeeds, the return value is DSJE_NOERROR and the event structure contains the details of the requested event.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

DSJE_BADEVENTID
Invalid event if for a specified job.

Remarks

Entries in the log file are numbered sequentially starting from 0. The latest event ID can be obtained through a call to **DSGetNewestLogId**. When a log is cleared, there always remains a single entry saying when the log was cleared.

DSGetLogEntryFull

Retrieves detailed information about a specific entry in a job log, including the message ID and the invocation ID.

Syntax

```
int DSGetLogEntryFull(
     DSJOB JobHandle,
     int EventId,
     DSLOGDETAILFULL *Event
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

EventId is the identifier for the event to be retrieved, see "Remarks."

Event is a pointer to a data structure to hold details of the log entry.

Return Values

If the function succeeds, the return value is DSJE_NOERROR and the event structure contains the details of the requested event.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

DSJE_BADEVENTID
Invalid event if for a specified job.

Remarks

Entries in the log file are numbered sequentially starting from 0. The latest event ID can be obtained through a call to **DSGetNewestLogId**. When a log is cleared, there always remains a single entry saying when the log was cleared.

DSGetLogEventIds

Retrieves a list of event log IDs for a given job invocation.

Syntax

```
int DSGetLogEventIds(
      DSJOB JobHandle,
      int RunNumber,
      char *Filter,
      char **List);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

RunNumber identifies the job invocation run for which event IDs are returned. Usually a zero value requests IDs for the most recent run of the job invocation. To retrieve details for earlier runs, supply negative values, such as -1 for details about the run before the most recent, -2 for details about the run before that, and so

forth. Where explicit run numbers are known, you can retrieve details by supplying the run number as a positive value.

Filter restricts the types of event log entry for which IDs are returned. By default, IDs for all log event entries are returned. Include characters in the filter string to restrict entries as follows:

I Informational

W Warning

F Fatal

S Start or End events

B Batch or Control events

R Purge or reset events

J Reject events

List is a pointer to a character buffer that receives the returned ID list.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid job handle.

DSJE_BADTYPE
Invalid *Filter* value.

DSJE_BADVALUE
Invalid *RunNumber* value.

Remarks

To use this method, the program needs to have previously acquired a job handle by calling **DSOpenJob()**.

The run number for a job invocation is reset when the job is compiled, thus it is not possible to use this method to retrieve job event IDs for runs that occurred prior to the most recent job compilation.

DSGetNewestLogId

Use the DSGetNewestLogId function to obtain the identifier of the newest entry in the jobs log.

Syntax

```
int DSGetNewestLogId(
   DSJOB JobHandle,
  int EventType
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

EventType is a key specifying the type of log entry whose identifier you want to retrieve and can be one of the following:

This key...
Retrieves this type of log entry...

DSJ_LOGINFO
Information

DSJ_LOGWARNING
Warning

DSJ_LOGFATAL
Fatal

DSJ_LOGREJECT
Transformer row rejection

DSJ_LOGSTARTED
Job started

DSJ_LOGRESET
Job reset

DSJ_LOGOTHER
Any other log event type

DSJ_LOGBATCH
Batch control

DSJ_LOGANY
Any type of event

Return Values

If the function succeeds, the return value is the positive identifier of the most recent entry of the requested type in the job log file.

If the function fails, the return value is -1. Use **DSGetLastError** to retrieve one of the following error codes:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

DSJE_BADTYPE
Invalid *EventType* value.

Remarks

Use this function to determine the ID of the latest entry in a log file before starting a job run. After the job has started or finished, it is then possible to determine which entries have been added by the job run.

DSGetParamInfo

Use the DSGetParamInfo function to retrieve information about a particular parameter within a job.

Syntax

```
int DSGetParamInfo(   DSJOB JobHandle,   char *ParamName,   DSPARAMINFO *ReturnInfo
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

ParamName is a pointer to a null-terminated string specifying the name of the parameter to be interrogated.

ReturnInfo is a pointer to a **DSPARAMINFO** data structure where the requested information is stored. For more information, see "Data Structures" on page 75.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

DSJE_BADHANDLE
Invalid *JobHandle.*

Remarks

Unlike the other information retrieval functions, **DSGetParamInfo** returns all the information relating to the specified item in a single call. The **DSPARAMINFO** data structure contains all the information required to request a new parameter value from a user and partially validate it. See "Data Structures" on page 75.

This function can be used either before or after a **DSRunJob** call has been issued:

- If called after a successful call to **DSRunJob**, the information retrieved refers to that run of the job.
- If called before a call to **DSRunJob**, the information retrieved refers to any previous run of the job, and not to any call to **DSSetParam** that might have been issued.

DSGetProjectInfo

Use the DSGetProjectInfo function to obtain a list of jobs in a project.

Syntax

```
int DSGetProjectInfo(
    DSPROJECT ProjectHandle,
    int InfoType,
    DSPROJECTINFO *ReturnInfo
);
```

Parameters

ProjectHandle is the value returned from **DSOpenProject**.

InfoType is a key indicating the information to be returned.

This key...
Retrieves this type of log entry...

DSJ_JOBLIST
Lists all jobs within the project.

DSJ_PROJECTNAME
Name of current project.

DSJ_HOSTNAME
Host name of the engine tier.

ReturnInfo is a pointer to a **DSPROJECTINFO** data structure where the requested information is stored.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There are no compiled jobs defined within the project.

DSJE_BADTYPE
Invalid *InfoType.*

Remarks

The **DSPROJECTINFO** data structure contains a **union** with an element for each of the possible return values from a call to **DSGetProjectInfo**.

Note: The returned list contains the names of all jobs known to the project, whether they can be opened or not.

.

DSGetProjectList

Use the DSGetProjectList function to obtain a list of all projects on the host system.

Syntax

```
char* DSGetProjectList(void);
```

Return Values

If the function succeeds, the return value is a pointer to a series of null-terminated strings, one for each project on the host system, ending with a second null character. The following example shows the buffer contents with <null> representing the terminating null character:

```
project1<null>project2<null><null>
```

If the function fails, the return value is NULL. And the **DSGetLastError** function retrieves the following error code:

DSJE_SERVER_ERROR Unexpected/unknown engine error occurred.

Remarks

This function can be called before any other InfoSphere DataStage API function.

Note: DSGetProjectList opens, uses, and closes its own communications link with the engine, so it might take some time to retrieve the project list.

DSGetReposInfo

Use the DSGetReposInfo function to search for design-time objects.

Syntax

```
int DSGetReposInfo (
    DSPROJECThProject,
    int ObjectType,
    int InfoType,
    const char *SearchCriteria,
    const char *StartingCategory,
    int Subcategories,
    DSREPOSINFO *ReturnInfo);
```

Parameters

hProject is the value returned from **DSOpenProject** for the project whose jobs you want to search.

ObjectType must currently be set to DSS_JOBS to indicate that you want to search for jobs.

InfoType is one or more of the following keys:

This key...
Returns this information...

DSS_JOB_ALL
Lists all jobs

DSS_JOB_SERVER
Lists all server jobs

DSS_JOB_PARALLEL
Lists all parallel jobs

DSS_JOB_MAINFRAME
Lists all mainframe jobs

DSS_JOB_SEQUENCE
Lists all job sequences

SearchCriteria is the name to match against. Partial name matching can be used, with multiple * characters used as wild cards anywhere in the search string.

StartingCategory is the category to start the search in. If no category name is supplied, or a NULL or empty string, then the root category is assumed.

SearchSubcategories can have one of two values: 1 (TRUE) and 0 (FALSE). These define whether the search is to include subcategories.

ReturnInfo is a pointer to a structure containing the required return information (see "DSREPOSINFO").

Return Value

On success, **DSGetReposInfo** returns the number of objects that have been found.

On failure an error code is returned as follows:

- DSJE_BADTYPE *ObjectType* or *InfoType* values was not recognized
- DSJE_REPERROR An error occurred while trying to access the repository. Call **DSGetLastErrorMsg** to get the error message associated with the error code
- DSJE_NO_DATASTAGE The attached project does not appear to be a valid InfoSphere DataStage project

DSGetReposUsage

Use the DSGetReposUsage function to return a list of objects based on the required relationship.

Syntax

```
int DSGetReposUsage(
   DSPROJECT hProject,
   int RelationshipType,
   const char *ObjectName,
   int Recursive,
   DSREPOSUSAGE *ReturnInfo);
```

Parameters

hProject is the value returned from **DSOpenProject for the project whose jobs you want to search.**

RelationshipType is one of the following keys:

This key...
Returns this information...

DSS_JOB_USES_JOB
Return a list of jobs that the specified job uses.

DSS_JOB_USEDBY_JOB
Return a list of jobs the specified job is used by.

DSS_JOB_HASSOURCE_ DRSTABLE
Return a list of jobs that use the specified table in as a source in a DRS Stage.

DSS_JOB_HASTARGET_ DRSTABLE
Return a list of jobs that use the specified table as a target in a DRS Stage.

DSS_JOB_HASSOURCEORTARGET_DRSTABLE
Returns a list of jobs that use the specified table as a source or target of a DRS Stage.

ObjectName specifies the job or table, and varies according to which *RelationshipType* is specified:

- for DSS_JOB_USES_JOB and DSS_JOB_USEDBY_JOB relationships, the job name (without category qualification) should be given.
- for remaining relationships, the fully qualified table name should be given.
- For the DRS Stage table definition relationships, partial matching of the table name using * characters as wild cards is allowed. Multiple wildcard characters can be used

Recursive is used by the DSS_JOB_USES_JOB and DSS_JOB_USEDBY_JOB relationships. It can have 2 values, 1 (TRUE) and 0 (FALSE). If set to TRUE, then for each job found that uses the *ObjectName*, the jobs which that job is used in are found and so on. For all other relationship types the parameter is ignored.

ReturnInfo is a pointer to a structure containing the returned values (see"DSREPOSUSAGE" on page 87). The order in which jobs appear in the *ReturnInfo* structure is defined by the *RelationshipType*. For the DSS_JOB_USES_JOB *RelationshipType*, the jobs will appear in the order in which they appear in the jobs dependency list. This list is on the **Dependencies** tab on the Job Properties dialog.

Return Value

On success, **DSGetReposUsage** returns the number of objects that have been found.

On failure an error code is returned as follows:

- DSJE_REPERROR An error occurred while trying to access the repository. Call n **DSGetLastErrorMsg** to get the error message associated with the error code.
- DSJE_NO_DATASTAGE The attached project does not appear to be a valid InfoSphere DataStage project.
- DSJE_UNKNOWN_JOBNAME When the *RelationshipType* is DSS_JOB_USES_JOB or DSS_JOB_USEDBY_JOB the supplied job name cannot be found in the project.

DSGetStageInfo

Use the DSGetStageInfo function to obtain information about a particular stage within a job.

Syntax

```
int DSGetStageInfo(
   DSJOB JobHandle,
   char *StageName,
   int InfoType,
   DSSTAGEINFO *ReturnInfo
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

StageName is a pointer to a null-terminated string specifying the name of the stage to be interrogated.

InfoType is one of the following keys:

This key...
Returns this information...

DSJ_LINKLIST
Null-separated list of names of links in stage.

DSJ_STAGELASTERR
Last error message reported from any link of the stage.

DSJ_STAGENAME
Stage name.

DSJ_STAGETYPE
Stage type name.

DSJ_STAGEINROWNUM
Primary links input row number.

DSJ_VARLIST
Null-separated list of stage variable names in the stage.

DSJ_STAGESTARTTIMESTAMP
Date and time when stage started.

DSJ_STAGEENDTIMESTAMP
Date and time when stage finished.

DSJ_STAGEDESC
Stage description (from stage properties)

DSJ_STAGEINST
Null-separated list of instance ids (parallel jobs).

DSJ_STAGECPU
List of CPU time in seconds.

DSJ_LINKTYPES
Null-separated list of link types.

DSJ_STAGEELAPSED
Elapsed time in seconds.

DSJ_STAGEPID
Null-separated list of process ids.

DSJ_STAGESTATUS
Stage status.

DSJ_CUSTINFOLIST
Null-separated list of custinfo items.

ReturnInfo is a pointer to a **DSSTAGEINFO** data structure where the requested information is stored. See "Data Structures" on page 75.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There are no instances of the requested information in the stage.

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADSTAGE
StageName does not refer to a known stage in the job.

DSJE_BADTYPE
Invalid *InfoType*.

Remarks

This function can be used either before or after a **DSRunJob** function has been issued.

The **DSSTAGEINFO** data structure contains a **union** with an element for each of the possible return values from the call to **DSGetStageInfo**.

DSGetVarInfo

Use the DSGetVarInfo function to obtain information about variables used in Transformer stages.

Syntax

```
int DSGetVarInfo(
   DSJOB JobHandle,
   char *StageName,
   char *VarName
   int InfoType,
   DSSTAGEINFO *ReturnInfo
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

StageName is a pointer to a null-terminated string specifying the name of the stage to be interrogated.

VarName is a pointer to a null-terminated string specifying the name of the variable to be interrogated.

InfoType is one of the following keys:

This key...
Returns this information...

DSJ_VARVALUE
The value of the specified variable.

DSJ_VARDESC
Description of the variable.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_NOT_AVAILABLE
There are no instances of the requested information in the stage.

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADSTAGE
StageName does not refer to a known stage in the job.

DSJE_BADVAR
VarName does not refer to a known variable in the job.

DSJE_BADTYPE
Invalid *InfoType*.

DSListEnvVars

Use the DSListEnvVars function to obtain a list of environment variables and their values in a specified project.

Syntax

```
char *DSListEnvVars(
      DSPROJECT hProject);
```

Parameter

hProject is the value returned from **DSOpenProject** for the project whose environment variables you want to list.

Return Values

If the function succeeds, the return value is a pointer to a series of null-terminated strings, one for each environment variable, ending with a second null character. Each string is of the format *EnvVarName=EnvVarValue*.

If the function fails, the return value is NULL and the DSGetLastError function can be used to retrieve an error code as follows:

- DSJE_READENVVARDEFNS failed to read environment variable definitions
- DSJE_READENVVARVALUES failed to read environment variable values
- DSJE.ISPARALLELLICENCED failed to determine if parallel jobs are available

Remarks

To use this method, the program needs to have previously attached to a project using **DSOpenProject**. This returns a handle to the project, *hProject*.

Environment variables in the parallel category will only be listed if parallel jobs are available.

DSListProjectProperties

Use the DSListProjectProperties function to obtain a list of the values of project properties for a specified project.

The properties supported by the DSListProjectProperties function are:

- Whether generated OSH is visible in parallel jobs.
- Whether runtime column propagation is enabled in parallel jobs.
- The base directory name for parallel jobs.
- Advanced runtime options for parallel jobs.
- Custom deployment commands for parallel jobs.
- Deployment job template directory.
- Whether job administration is enabled in the Director client or not.

Syntax

```
char *DSListProjectProperties(
      DSPROJECT hProject
);
```

Parameter

hProject is the value returned from **DSOpenProject for the project whose properties you want to list.**

Return Values

If the function succeeds, the return value is a pointer to a series of null-terminated strings, one for each variable, ending with a second null character. Each string is of the format *PropertyName=PropertyValue* where *PropertyName* will be one of the following:

This key...
Indicates this property...

DSA_OSHVISIBLEFLAG
Generated OSH is visible in parallel jobs. Parallel jobs only.

DSA_PRJ_RTCP_ENABLED
Runtime column propagation is enabled in parallel jobs. Parallel jobs only.

DSA_PRJ_PX_ADVANCED_RUNTIME_OPTS
Specifies advanced runtime properties for parallel jobs. Parallel jobs only.

DSA_PRJ_PX_BASEDIR
Specifies the base directory for parallel jobs. Parallel jobs only.

DSA_PRJ_PX_DEPLOY_JOBDIR_TEMPLATE
Specifies the deployment directory template for parallel jobs. Parallel jobs only.

DSA_PRJ_PX_DEPLOY_CUSTOM_ACTION
Specifies custom deployment commands for parallel jobs. *Value* is the commands. Parallel jobs only.

DSA_PRJ_JOBADMIN_ENABLED
Job administration commands are enabled in the Director client for jobs in this project.

DSA_PRJ_PX_DEPLOY_GENERATE_XML
Generation of XML reports is enabled for parallel job deployment packages.

These tokens are defined in *dsapi.h* (see "The dsapi.h Header File" on page 35).

If the function fails, the return value is NULL and the **DSGetLastError** function can be used to retrieve one of the following error code:

- DSJE_READPROJPROPERTY failed to read property
- DSJE_ISPARALLELLICENCED failed to determine if parallel jobs are available
- DSJE_OSHVISIBLEFLAG failed to get value for OSHVisible

Remarks

To use this method, the program needs to have previously attached to a project using **DSOpenProject**. This returns a handle to the project, *hProject*.

If parallel jobs are not available, only the setting of the DSA_PRJ_JOBADMIN_ENABLED will be returned.

DSLockJob

Use the DSLockJob function to lock a job. You must call this function before you set a job's run parameters or start a job run.

Syntax

```
int DSLockJob(
   DSJOB JobHandle
);
```

Parameter

JobHandle is the value returned from **DSOpenJob**.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

Remarks

Locking a job prevents any other process from modifying the job details or status. This function must be called before any call of **DSSetJobLimit**, **DSSetParam**, or **DSRunJob**.

If you try to lock a job you already have locked, the call succeeds. If you have the same job open on several InfoSphere DataStage API handles, locking the job on one handle locks the job on all the handles.

DSLogEvent

Use the DSLogEvent function to add a new entry to a job log file.

Syntax

```
int DSLogEvent(
   DSJOB JobHandle,
   int EventType,
   char *Reserved,
   char *Message
);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

EventType is one of the following keys specifying the type of event to be logged:

This key...
Specifies this type of event...

DSJ_LOGINFO
Information

DSJ_LOGWARNING
Warning

Reserved is reserved for future use, and should be specified as null.

Message points to a null-terminated character string specifying the text of the message to be logged.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

DSJE_BADTYPE
Invalid *EventType* value.

Remarks

Messages that contain more that one line of text should contain a newline character (\n) to indicate the end of a line.

DSMakeJobReport

Use the DSMakeJobReport function to generate a report describing the complete status of a valid attached job.

Syntax

```
int DSMakeJobReport(
   DSJOB JobHandle,
   int ReportType,
   char *LineSeparator,
   DSREPORTINFO *ReturnInfo);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

ReportType is one of the following values specifying the type of report to be generated:

This value...
Specifies this type of report...

0 Basic, text string containing start/end time, time elapsed and status of job.

1 Stage/link detail. As basic report, but also contains information about individual stages and links within the job.

2 Text string containing full XML report.

LineSeparator points to a null-terminated character string specifying the line separator in the report. Special values recognized are:

"CRLF" => CHAR(13):CHAR(10)

"LF" => CHAR(10)

"CR" => CHAR(13)

The default is CRLF if on Windows, else LF.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

DSOpenJob

Use the DSOpenJob function to open a job. You must call this function before any other function that manipulates the job.

Syntax

```
DSJOB DSOpenJob(
   DSPROJECT ProjectHandle,
   char *JobName
);
```

Parameters

ProjectHandle is the value returned from **DSOpenProject**.

JobName is a pointer to a null-terminated string that specifies the name of the job that is to be opened. This might be in either of the following formats:

job Finds the latest version of the job.

*job***%Rel***n.n.n*
Finds a particular release of the job on a development system.

Return Values

If the function succeeds, the return value is a handle to the job.

If the function fails, the return value is NULL. Use **DSGetLastError** to retrieve one of the following:

Token Description

DSJE_OPENFAIL
Engine failed to open job.

DSJE_NO_MEMORY
Memory allocation failure.

Remarks

The **DSOpenJob** function must be used to return a job handle before a job can be addressed by any of the InfoSphere DataStage API functions. You can gain exclusive access to the job by locking it with **DSLockJob**.

The same job can be opened more than once and each call to **DSOpenJob** will return a unique job handle. Each handle must be separately closed.

DSOpenProject

Use the DSOpenProject function to open a project. You must call this function before any other InfoSphere DataStage API function, except the DSGetProjectList function or the DSGetLastError function.

Syntax

```
DSPROJECT DSOpenProject(
   char *ProjectName
);
```

Parameter

ProjectName is a pointer to a null-terminated string that specifies the name of the project to open.

Return Values

If the function succeeds, the return value is a handle to the project.

If the function fails, the return value is NULL. Use **DSGetLastError** to retrieve one of the following:

Token Description

DSJE_BAD_VERSION
The engine is an older version than the InfoSphere DataStage API.

DSJE_INCOMPATIBLE_SERVER
The engine is either older or newer than that supported by this version of InfoSphere DataStage API.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

DSJE_BADPROJECT
Invalid project name.

DSJE_NO_DATASTAGE
InfoSphere DataStage is not correctly installed on engine tier host.

Remarks

The **DSGetProjectList** function can return the name of a project that does not contain valid InfoSphere DataStage jobs, but this is detected when **DSOpenProject** is called. A process can only have one project open at a time.

DSRunJob

Use the DSRunJob function to start a job run.

Syntax

```
int DSRunJob(   DSJOB JobHandle,
      int RunMode
);
```

Parameters

JobHandle is a value returned from **DSOpenJob**.

RunMode is a key determining the run mode and should be one of the following values:

DSJ_RUNNORMAL
Start a job run.

DSJ_RUNRESET
Reset the job.

DSJ_RUNVALIDATE
Validate the job.

DSJ_RUNRESTART
Restart a restartable job sequence with the original job parameter values.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following tokens:

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADSTATE
Job is not in the right state (must be compiled and not running).

DSJE_BADTYPE
RunMode is not recognized.

DSJE_SERVER_ERROR
Internal error. The engine returned invalid data.

Remarks

The job specified by *JobHandle* must be locked, using **DSLockJob**, before the **DSRunJob** function is called.

If no limits were set by calling **DSSetJobLimit**, the default limits are used.

DSSetEnvVar

Use the SDSetEnvVar function to set the value for an environment variable in a specified project.

Syntax

```
int DSSetEnvVar(
      DSPROJECT hProject,
      char *EnvVarName,
      char *Value
);
```

Parameters

hProject is the value returned from **DSOpenProject**.

EnvVarName is the name of the environment variable.

Value is the value to set the environment variable to.

Return Values

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_READENVVARDEFNS failed to read environment variable definitions
- DSJE_READENVVARVALUES failed to read environment variable values
- DSJE_BADENVVAR environment variable does not exist
- DSJE_WRITEENVVARVALUES failed to write environment variable values
- DSJE_ENCODEFAILED failed to encode an encrypted value
- DSJE_BADBOOLEANVALUE invalid value given for a boolean environment variable
- DSJE_BADNUMERICVAL UE invalid value given for an integer environment variable
- DSJE_BADLISTVALUE invalid value given for an environment variable with a fixed list of values
- DSJE_PXNOTINSTALLED environment variable is specific to parallel jobs which are not available
- DSJE_ISPARALLELLICENCED failed to determine if parallel jobs are available

Remarks

You can only set values for environment variables in the parallel category if parallel jobs are available.

If setting a list type environment variable (for example, APT_EXECUTION_MODE), then you should set it to one of the permissible internal values, rather than one of the list members as they are shown in the Administrator client. For example, if you wanted to set APT_EXECUTION_MODE so that parallel jobs executed in one process mode, you would set the environment variable value to `ONE_PROCESS', not `One process' as offered in the Administrator client.

If you are setting a boolean type environment variable, set the value to 1 for TRUE and 0 for FALSE.

DSSetGenerateOpMetaData

Use the DSSetGenerateOpMetaData function to specify whether the job generates operational metadata or not. This function overrides the default setting for the project.

Syntax

```
int DSSetGenerateOpMetaData (
   JobHandle,
   value
);
```

Parameters

JobHandle is a value returned from **DSOpenJob**.

value is TRUE (1) to generate operational metadata, FALSE (0) to not generate operational metadata.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADTYPE
value is not recognized.

DSSetJobLimit

Use the DSSetJobLimit function to set row or warning limits for a job.

Syntax

```
int DSSetJobLimit(
   DSJOB JobHandle,
   int LimitType,
   int LimitValue
);
```

Parameters

JobHandle is a value returned from **DSOpenJob**.

LimitType is one of the following keys specifying the type of limit:

This key...
Specifies this type of limit...

DSJ_LIMITWARN
Job to be stopped after *LimitValue* warning events.

DSJ_LIMITROWS
Stages to be limited to *LimitValue* rows.

LimitValue is the value to set the limit to.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_BADSTATE
Job is not in the right state (compiled, not running).

DSJE_BADTYPE
LimitType is not the name of a known limiting condition.

DSJE_BADVALUE
LimitValue is not appropriate for the limiting condition type.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

Remarks

The job specified by *JobHandle* must be locked, using **DSLockJob**, before the **DSSetJobLimit** function is called.

Any job limits that are not set explicitly before a run will use the default values. Make two calls to **DSSetJobLimit** in order to set both types of limit.

Set the value to 0 to indicate that there should be no limit for the job.

DSSetParam

Use the DSSetParam function to set job parameter values before you run a job. Any parameter that is not explicitly set uses the default value.

Syntax

```
int DSSetParam(
   DSJOB JobHandle,
   char *ParamName,
   DSPARAM *Param);
```

Parameters

JobHandle is the value returned from **DSOpenJob**.

ParamName is a pointer to a null-terminated string that specifies the name of the parameter to set.

Param is a pointer to a structure that specifies the name, type, and value of the parameter to set.

Note: The type specified in *Param* need not match the type specified for the parameter in the job definition, but it must be possible to convert it. For example, if the job defines the parameter as a string, it can be set by specifying it as an integer. However, it will cause an error with unpredictable results if the parameter is defined in the job as an integer and a nonnumeric string is passed by **DSSetParam**.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle*.

DSJE_BADSTATE
Job is not in the right state (compiled, not running).

DSJE_BADPARAM
Param does not reference a known parameter of the job.

DSJE_BADTYPE
Param does not specify a valid parameter type.

DSJE_BADVALUE
Param does not specify a value that is appropriate for the parameter type as specified in the job definition.

DSJE_SERVER_ERROR
Internal error. Engine returned invalid data.

Remarks

The job specified by *JobHandle* must be locked, using **DSLockJob**, before the **DSSetParam** function is called.

DSSetProjectProperty

Use the DSSetProjectProperty function to set the value of a property in a specified project. The user who runs the code that contains this function must be an InfoSphere DataStage administrator.

Syntax

```
int DSSetProjectProperty(
      DSPROJECT hProject,
      char *Property,
      char *Value
);
```

Parameters

hProject is the value returned from **DSOpenProject**

Property is the name of the property to set. The following properties are supported:

This key...
Indicates this property...

DSA_OSHVISIBLEFLAG
Generated OSH is visible in parallel jobs, *Value* is 0 for false or 1 for true. Parallel jobs only.

DSA_PRJ_RTCP_ENABLED
Runtime column propagation is enabled in parallel jobs, *Value* is 0 for false or 1 for true. Parallel jobs only.

DSA_PRJ_PX_ADVANCED_RUNTIME_OPTS
Specifies advanced runtime properties for parallel jobs, *Value* is the advanced properties to set. Parallel jobs only.

DSA_PRJ_PX_BASEDIR
Specifies the base directory for parallel jobs. *Value* is the base directory. Parallel jobs only.

DSA_PRJ_PX_DEPLOY_JOBDIR_TEMPLATE
Specifies the deployment directory template for parallel jobs. *Value* is the deployment directory template. Parallel jobs only.

DSA_PRJ_PX_DEPLOY_CUSTOM_ACTION
Specifies custom deployment commands for parallel jobs. *Value* is the commands. Parallel jobs only.

DSA_PRJ_JOBADMIN_ENABLED
Job administration commands are enabled in the Director client for jobs in this project. *Value* is 0 for false or 1 for true.

DSA_PRJ_PX_DEPLOY_GENERATE_XML
Generation of XML reports is enabled for Parallel job deployment packages. *Value* is 0 for false or 1 for true.

Return Values

If the function succeeds, then the return value is DSJE_NOERROR

If the function fails, then the return value is one of the following:

- DSJE_NOTADMINUSER user is not an administrator
- DSJE_ISADMINFAILED failed to determine whether user is an administrator
- DSJE_READPROJPROPERTY failed to read property
- DSJE_WRITEPROJPROPERTY failed to write property
- DSJE_PROPNOTSUPPORTED property not supported
- DSJE_BADPROPERTY unknown property name
- DSJE_BADPROPVALUE invalid value for this property
- DSJE_PXNOTINSTALLED parallel jobs not available
- DSJE_ISPARALLELLICENCED failed to determine if parallel jobs are available
- DSJE_OSHVISIBLEFLAG failed to set value for OSHVisible

Remarks

To use this method, the program needs to have previously attached to a project using **DSOpenProject**. This returns a handle to the project, *hProject*.

DSSetServerParams

Use the DSSetServerParams function to set the logon parameters to use for opening a project or retrieving a project list.

Syntax

```
void DSSetServerParams(
      char *DomainName,
      char *UserName,
      char *Password,
      char *ServerName
);
```

Parameters

DomainName is a pointer to a null-terminated character string specifying the name of the domain machine.

UserName is a pointer to either a null-terminated character string specifying the user name to use for the session, or NULL.

Password is a pointer to either a null-terminated character string specifying the password for the user specified in *UserName*, or NULL.

ServerName is a pointer to either a null-terminated character string specifying the name of the engine tier server to connect to, or NULL.

Return Values

This function has no return value.

Remarks

By default, **DSOpenProject** and **DSGetProjectList** attempt to connect to an engine on the same computer as the client process, then create an engine process that runs with the same user identification and access rights as the client process. **DSSetServerParams** overrides this behavior and allows you to specify a different domain, user name, password, and engine tier server.

Calls to **DSSetServerParams** are not cumulative. All parameter values, including NULL pointers, are used to set the parameters to be used on the subsequent **DSOpenProject** or **DSGetProjectList** call.

DSStopJob

Use the DSStopJob to cancel a running job.

Syntax

```
int DSStopJob(
   DSJOB JobHandle
);
```

Parameter

JobHandle is the value returned from **DSOpenJob**.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is:

DSJE_BADHANDLE Invalid *JobHandle.*

Remarks

The **DSStopJob** function should be used only after a **DSRunJob** function has been issued. The stop request is sent regardless of the job's current status. To ascertain if the job has stopped, use the **DSWaitForJob** function or the **DSJobStatus** macro.

DSUnlockJob

Use the DSUnlockJob function to unlock a job, preventing any further manipulation of the job's run state and freeing it for other processes to use.

Syntax

```
int DSUnlockJob(
   DSJOB JobHandle
);
```

Parameter

JobHandle is the value returned from **DSOpenJob**.

Return Values

If the function succeeds, the return value is DSJ_NOERROR.

If the function fails, the return value is:

- DSJE_BADHANDLE Invalid *JobHandle.*

Remarks

The **DSUnlockJob** function returns immediately without waiting for the job to finish. Attempting to unlock a job that is not locked does not cause an error. If you have the same job open on several handles, unlocking the job on one handle unlocks it on all handles.

DSWaitForJob

Use the DSWaitForJob function to wait to the completion of a job run.

Syntax

```
int DSWaitForJob(
   DSJOB JobHandle
);
```

Parameter

JobHandle is the value returned from **DSOpenJob**.

Return Values

If the function succeeds, the return value is DSJE_NOERROR.

If the function fails, the return value is one of the following:

Token Description

DSJE_BADHANDLE
Invalid *JobHandle.*

DSJE_WRONGJOB
Job for this *JobHandle* was not started from a call to **DSRunJob** by the current process.

DSJE_TIMEOUT
Job appears not to have started after waiting for a reasonable length of time. (About 30 minutes.)

Remarks

This function is only valid if the current job has issued a **DSRunJob** call on the given *JobHandle*. It returns if the job was started since the last **DSRunJob**, and has since finished. The finishing status can be found by calling **DSGetJobInfo**.

Data Structures

Use data structures to hold data passed to, or returned from, functions.

The InfoSphere DataStage API uses the data structures described in this section to hold data passed to, or returned from, functions. (See "Data Structures, Result Data, and Threads" on page 36). The data structures are summarized below, with full descriptions in the following sections:

This data structure...	Holds this type of data...	And is used by this function...
DSCUSTINFO	Custinfo items from certain types of parallel stage	DSGetCustinfo
DSJOBINFO	Information about a job	DSGetJobInfo
DSLINKINFO	Information about a link to or from an active stage in a job, that is, a stage that is not a data source or destination	DSGetLinkInfo
DSLOGDETAIL	Full details of an entry in a job log file	DSGetLogEntry
DSLOGDETAILFULL	Full details of an entry in a job log file, including the message ID and the invocation ID	DSGetLogEntryFull
DSLOGEVENT	Details of an entry in a job log file	DSLogEvent, DSFindFirstLogEntry, DSFindNextLogEntry
DSPARAM	The type and value of a job parameter	DSSetParam
DSPARAMINFO	Further information about a job parameter, such as its default value and a description	DSGetParamInfo
DSPROJECTINFO	A list of jobs in the project	DSGetProjectInfo
DSREPOSINFO	A list of design time jobs	DSGetReposInfo
DSREPOSUSGE	A list of design time jobs satisfying a relationship	DSGetReposUsage
DSSTAGEINFO	Information about a stage in a job	DSGetStageInfo
DSVARINFO	Information about stage variables in transformer stages	DSGetVarInfo

DSCUSTINFO

Use the **DSCUSTINFO** structure to represent various information values about a link to or from an active stage within an InfoSphere DataStage job.

Syntax

```
typedef struct _DSCUSTINFO {
int infoType:   /
union {
   char *custinfoValue;
   char *custinfoDesc;} info;
} DSCUSTINFO;
```

Members

infoType is a key indicating the type of information and is one of the following values:

This key...
Indicates this information...

DSJ_CUSTINFOVALUE
The value of the specified custinfo item.

DSJ_CUSTINFODESC
The description of the specified custinfo item.

DSJOBINFO

Use the DSJOBINFO structure to represent information values about an InfoSphere DataStage job.

Syntax

```
typedef struct _DSJOBINFO {
   int infoType;
   union {
      int jobStatus;
      char *jobController;
      time_t jobStartTime;
      int jobWaveNumber;
      char *userStatus;
      char *paramList;
      char *stageList;
      char *jobname;
      int jobcontrol;
      int jobPid;
      time_t jobLastTime;
      char *jobInvocations;
      int jobInterimStatus;
      char *jobInvocationid;
      char *jobDesc;
      char *stageList2;
      char *jobElapsed;
      char *jobFullDesc;
      int jobDMIService;
      int jobMultiInvokable;
   } info;
} DSJOBINFO;
```

Members

infoType is one of the following keys indicating the type of information:

This key...
Indicates this information...

DSJ_JOBSTATUS
The current status of the job.

DSJ_JOBNAME
Name of job referenced by *JobHandle*

DSJ_JOBCONTROLLER
The name of the controlling job.

DSJ_JOBSTARTTIMESTAMP
The date and time when the job started.

DSJ_JOBWAVENO
Wave number of the current (or last) job run.

DSJ_PARAMLIST
A list of the names of the job's parameters. Separated by nulls.

DSJ_STAGELIST
A list of active stages in the job. Separated by nulls.

DSJ_USERSTATUS
The status reported by the job itself as defined in the job's design.

DSJ_JOBCONTROL
Whether a stop request has been issued for the job.

DSJ_JOBPID
Process id of DSD.RUN process.

DSJ_JOBLASTTIMESTAMP
The date and time on the engine when the job last finished.

DSJ_JOBINVOCATIONS
List of job invocation ids. Separated by nulls.

DSJ_JOBINTERIMSTATUS
Current Interim status of the job.

DSJ_JOBINVOVATIONID
Invocation name of the job referenced.

DSJ_JOBDESC
A description of the job.

DSJ_STAGELIST2
A list of passive stages in the job. Separated by nulls.

DSJ_JOBELAPSED
The elapsed time of the job in seconds.

DSJ_JOBFULLDESSC
The Full Description specified in the Job Properties dialog box.

DSJ_JOBDMISERVICE
Set to true if this is a Web service job.

DSJ_JOBMULTIINVOKABLE
Set to true if this job supports multiple invocations.

jobStatus is returned when *infoType* is set to DSJ_JOBSTATUS. Its value is one of the following keys:

This key...
Indicates this status...

DSJS_RUNNING
Job running.

DSJS_RUNOK
Job finished a normal run with no warnings.

DSJS_RUNWARN
Job finished a normal run with warnings.

DSJS_RUNFAILED
Job finished a normal run with a fatal error.

DSJS_VALOK
Job finished a validation run with no warnings.

DSJS_VALWARN
Job finished a validation run with warnings.

DSJS_VALFAILED
Job failed a validation run.

DSJS_RESET
Job finished a reset run.

DSJS_CRASHED
Job was stopped by some indeterminate action.

DSJS_STOPPED
Job was stopped by operator intervention (can't tell run type).

DSJS_NOTRUNNABLE
Job has not been compiled.

DSJS_NOTRUNNING
Any other status. Job was stopped by operator intervention (can't tell run type).

jobController is the name of the job controlling the job reference and is returned when *infoType* is set to DSJ_JOBCONTROLLER. Note that this can be several job names, separated by periods, if the job is controlled by a job which is itself controlled, and so on.

jobStartTime is the date and time when the last or current job run started and is returned when *infoType* is set to DSJ_JOBSTARTTIMESTAMP.

jobWaveNumber is the wave number of the last or current job run and is returned when *infoType* is set to DSJ_JOBWAVENO.

userStatus is the value, if any, set by the job as its user defined status, and is returned when *infoType* is set to DSJ_USERSTATUS.

paramList is a pointer to a buffer that contains a series of null-terminated strings, one for each job parameter name, that ends with a second null character. It is returned when *infoType* is set to DSJ_PARAMLIST. The following example shows the buffer contents with <null> representing the terminating null character:

```
first<null>second<null><null>
```

stageList is a pointer to a buffer that contains a series of null-terminated strings, one for each stage in the job, that ends with a second null character. It is returned when *infoType* is set to DSJ_STAGELIST. The following example shows the buffer contents with <null> representing the terminating null character:

```
first<null>second<null><null>
```

DSLINKINFO

Use the DSLINKINFO structure to represent various information values about a link to or from an active stage within an InfoSphere DataStage job.

Syntax

```
typedef struct _DSLINKINFO {
int infoType:   /
union {
```

```
   DSLOGDETAIL lastError;
   int rowCount;
   char *linkName;
   char *linkSQLState;
   char *linkDBMSCode;
   char *linkDesc;
   char *linkedStage;
   char *rowCountList;
} info;
} DSLINKINFO;
```

Members

infoType is a key indicating the type of information and is one of the following values:

This key...
Indicates this information...

DSJ_LINKLASTERR
The last error message reported from a link.

DSJ_LINKNAME
Actual name of link.

DSJ_LINKROWCOUNT
The number of rows that have been passed down a link.

DSJ_LINKSQLSTATE
SQLSTATE value from last error message.

DSJ_LINKDBMSCODE
DBMSCODE value from last error message.

DSJ_LINKDESC
Description of the link.

DSJ_LINKSTAGE
Name of the stage at the other end of the link.

DSJ_INSTROWCOUNT
Comma-separated list of row counts, one per instance (parallel jobs)

lastError is a data structure containing the error log entry for the last error message reported from a link and is returned when *infoType* is set to DSJ_LINKLASTERR.

rowCount is the number of rows that have been passed down a link so far and is returned when *infoType* is set to DSJ_LINKROWCOUNT.

DSLOGDETAIL

The **DSLOGDETAIL** structure represents detailed information for a single entry from a job log file.

Syntax

```
typedef struct _DSLOGDETAIL {
   int eventId;
   time_t timestamp;
   int type;
   char *username;
   char *fullMessage;
} DSLOGDETAIL;
```

Members

eventId is a number, 0 or greater, that uniquely identifies the log entry for the job.

timestamp is the date and time at which the entry was added to the job log file.

type is a key indicating the type of the event, and is one of the following values:

This key...
Indicates this type of log entry...

DSJ_LOGINFO
Information

DSJ_LOGWARNING
Warning

DSJ_LOGFATAL
Fatal error

DSJ_LOGREJECT
Transformer row rejection

DSJ_LOGSTARTED
Job started

DSJ_LOGRESET
Job reset

DSJ_LOGBATCH
Batch control

DSJ_LOGOTHER
Any other type of log entry

username is the user information.

fullMessage is the full description of the log entry.

DSLOGDETAILFULL

The **DSLOGDETAILFULL** structure represents detailed information for a single entry from a job log file, including the message ID and the invocation ID.

Syntax

```
typedef struct _DSLOGDETAILFULL {
   int eventId;
   time_t timestamp;
   int type;
   char *username;
   char *fullMessage;
   char *messageId;
   char *invocationId;
} DSLOGDETAILFULL;
```

Members

eventId is a number, 0 or greater, that uniquely identifies the log entry for the job.

timestamp is the date and time at which the entry was added to the job log file.

type is a key indicating the type of the event, and is one of the following values:

This key...
Indicates this type of log entry...

DSJ_LOGINFO
Information

DSJ_LOGWARNING
Warning

DSJ_LOGFATAL
Fatal error

DSJ_LOGREJECT
Transformer row rejection

DSJ_LOGSTARTED
Job started

DSJ_LOGRESET
Job reset

DSJ_LOGBATCH
Batch control

DSJ_LOGOTHER
Any other type of log entry

username is the user information.

fullMessage is the full description of the log entry.

messageId is a string containing the message ID.

invocationId is a string containing the invocation ID, if one was used.

DSLOGEVENT

Use the DSLOGEVENT structure to represent the summary information for a single entry from a job's event log.

Syntax

```
typedef struct _DSLOGEVENT {
   int eventId;
   time_t timestamp;
   int type;
   char *message;
} DSLOGEVENT;
```

Members

eventId is a number, 0 or greater, that uniquely identifies the log entry for the job.

timestamp is the date and time at which the entry was added to the job log file.

type is a key indicating the type of the event, and is one of the following values:

This key...
Indicates this type of log entry...

DSJ_LOGINFO
Information

DSJ_LOGWARNING
Warning

DSJ_LOGFATAL
Fatal error

DSJ_LOGREJECT
Transformer row rejection

DSJ_LOGSTARTED
Job started

DSJ_LOGRESET
Job reset

DSJ_LOGBATCH
Batch control

DSJ_LOGOTHER
Any other type of log entry

message is the first line of the description of the log entry.

DSPARAM

Use the DSPARAM structure to represent information about the type and value of an InfoSphere DataStage job parameter.

Syntax

```
typedef struct _DSPARAM {
int paramType;
union {
   char *pString;
   char *pEncrypt;
   int pInt;
   float pFloat;
   char *pPath;
   char *pListValue;
   char *pDate;
   char *pTime;
} paramValue;
} DSPARAM;
```

Members

paramType is a key specifying the type of the job parameter. Possible values are as follows:

This key...
Indicates this type of parameter...

DSJ_PARAMTYPE_STRING
A character string.

DSJ_PARAMTYPE_ENCRYPTED
An encrypted character string (for example, a password).

DSJ_PARAMTYPE_INTEGER
An integer.

DSJ_PARAMTYPE_FLOAT
A floating-point number.

DSJ_PARAMTYPE_PATHNAME
A file system path name.

DDSJ_PARAMTYPE_LIST
A character string specifying one of the values from an enumerated list.

DDSJ_PARAMTYPE_DATE
A date in the format *YYYY-MM-DD*.

DSJ_PARAMTYPE_TIME
A time in the format *HH*:*MM*:*SS*.

pString is a null-terminated character string that is returned when *paramType* is set to DSJ_PARAMTYPE_STRING.

pEncrypt is a null-terminated character string that is returned when *paramType* is set to DSJ_PARAMTYPE_ENCRYPTED. The string should be in plain text form when passed to or from InfoSphere DataStage API where it is encrypted. The application using the InfoSphere DataStage API should present this type of parameter in a suitable display format, for example, an asterisk for each character of the string rather than the character itself.

pInt is an integer and is returned when *paramType* is set to DSJ_PARAMTYPE_INTEGER.

pFloat is a floating-point number and is returned when *paramType* is set to DSJ_PARAMTYPE_FLOAT.

pPath is a null-terminated character string specifying a file system path name and is returned when *paramType* is set to DSJ_PARAMTYPE_PATHNAME.

Note: This parameter does not need to specify a valid path name on the engine. Interpretation and validation of the path name is performed by the job.

pListValue is a null-terminated character string specifying one of the possible values from an enumerated list and is returned when *paramType* is set to DDSJ_PARAMTYPE_LIST.

pDate is a null-terminated character string specifying a date in the format *YYYY-MM-DD* and is returned when *paramType* is set to DSJ_PARAMTYPE_DATE.

pTime is a null-terminated character string specifying a time in the format *HH*:*MM*:*SS* and is returned when *paramType* is set to DSJ_PARAMTYPE_TIME.

DSPARAMINFO

Use the DSPARAMINFO structure to represent information values about a parameter of an InfoSphere DataStage job.

Syntax

```
typedef struct _DSPARAMINFO {
   DSPARAM defaultValue;
   char *helpText;
   char *paramPrompt;
   int paramType;
   DSPARAM desDefaultValue;
```

```
   char *listValues;
   char *desListValues;
   int promptAtRun;
} DSPARAMINFO;
```

Members

defaultValue is the default value, if any, for the parameter.

helpText is a description, if any, for the parameter.

paramPrompt is the prompt, if any, for the parameter.

paramType is a key specifying the type of the job parameter. Possible values are as follows:

This key...
Indicates this type of parameter...

DSJ_PARAMTYPE_STRING
A character string.

DSJ_PARAMTYPE_ENCRYPTED
An encrypted character string (for example, a password).

DSJ_PARAMTYPE_INTEGER
An integer.

DSJ_PARAMTYPE_FLOAT
A floating-point number.

DSJ_PARAMTYPE_PATHNAME
A file system path name.

DDSJ_PARAMTYPE_LIST
A character string specifying one of the values from an enumerated list.

DDSJ_PARAMTYPE_DATE
A date in the format *YYYY-MM-DD*.

DSJ_PARAMTYPE_TIME
A time in the format *HH*:*MM*:*SS*.

desDefaultValue is the default value set for the parameter by the job's designer.

Note: Default values can be changed by the InfoSphere DataStage administrator, so a value might not be the current value for the job.

listValues is a pointer to a buffer that receives a series of null-terminated strings, one for each valid string that can be used as the parameter value, ending with a second null character as shown in the following example (<null> represents the terminating null character):

```
first<null>second<null><null>
```

desListValues is a pointer to a buffer containing the default list of values set for the parameter by the job's designer. The buffer contains a series of null-terminated strings, one for each valid string that can be used as the parameter value, that ends with a second null character. The following example shows the buffer contents with <null> representing the terminating null character:

```
first<null>second<null><null>
```

Note: Default values can be changed by the InfoSphere DataStage administrator, so a value might not be the current value for the job.

promptAtRun is either 0 (**False**) or 1 (**True**). 1 indicates that the operator is prompted for a value for this parameter whenever the job is run; 0 indicates that there is no prompting.

DSPROJECTINFO

Use the DSPROJECTINFO structure to represents information values for an InfoSphere DataStage project.

Syntax

```
typedef struct _DSPROJECTINFO {
   int infoType;
   union {
   char *jobList;
   } info;
} DSPROJECTINFO;
```

Members

infoType is a key value indicating the type of information to retrieve. Possible values are as follows:

This key...
: **Indicates this information...**

DSJ_JOBLIST
: List of jobs in project.

DSJ_PROJECTNAME
: Name of current project.

DSJ_HOSTNAME
: Host name of the engine tier.

jobList is a pointer to a buffer that contains a series of null-terminated strings, one for each job in the project, and ending with a second null character, as shown in the following example (<null> represents the terminating null character):

```
first<null>second<null><null>
```

DSREPOSINFO

Use the DSREPOSINFO structure to obtain information about design-time objects that have been searched for.

Syntax

```
struct _DSREPOSJOBINFO;
typedef struct _DSREPOSJOBINFO DSREPOSJOBINFO;
struct _DSREPOSJOBINFO
{
      char* jobname;    /* Includes category */
      int jobtype;      /* InfoType constant */
      DSREPOSJOBINFO* nextjob; /* ptr next job or NULL */
};
typedef struct _DSREPOSINFO
{
      int infoType;
      union
```

```
      {
         DSREPOSJOBINFO* jobs; /*linkedlist of found jobs */
      } info;
} DSREPOSINFO;
```

Members

infoType is a key value indicating the type of information to retrieve. Possible values are as follows:

This key...
Indicates this information...

DSS_JOBS
List of jobs.

jobs is a pointer to a structure linked to another structure, or terminated with a null. There is one structure for each job returned. Each structure contains the job name (including category) and the job type as follows:

This key...
Returns this information...

DSS_JOB_SERVER
Server job

DSS_JOB_PARALLEL
Parallel job

DSS_JOB_MAINFRAME
Mainframe job

DSS_JOB_SEQUENCE
Job sequence

DSREPOSUSAGE

Use the DSREPOSUSAGE structure to obtain information about objects that meet a specified relationship.

Syntax

struct _DSREPOSUSAGEJOB; typedef struct _DSREPOSUSAGEJOB DSREPOSUSAGEJOB; struct _DSREPOSUSAGEJOB { char **jobname*; /* Job and cat name */ int *jobtype*; /* type of job */ DSREPOSUSAGEJOB **nextjob*; /* next sibling job */ DSREPOSUSAGEJOB **childjob*; }; typedef struct _DSREPOSUSAGE { int *infoType*; union { DSREPOSUSAGEJOB **jobs*; /*linkedlist of jobs*/ } *info* } DSREPOSUSAGE;

Members

infoType is a key value indicating the type of information to retrieve. Possible values are as follows:

This key...
Indicates this information...

DSS_JOBS
List of jobs.

jobs is a pointer to a structure linked to another structure, or terminated with a null. There is one structure for each job returned. Each structure contains the job name (including category) and the job type as follows:

This key...
Returns this information...

DSS_JOB_SERVER
Server job

DSS_JOB_PARALLEL
Parallel job

DSS_JOB_MAINFRAME
Mainframe job

DSS_JOB_SEQUENCE
Job sequence

DSSTAGEINFO

Use the DSSTAGEINFO structure to represent various information values about an active stage within an InfoSphere DataStage job.

Syntax

```
typedef struct _DSSTAGEINFO {
int infoType;
union {
   DSLOGDETAIL lastError;
   char *typeName;
   int inRowNum;
   char *linkList;
   char *stagename;
   char *varlist;
   char *stageStartTime;
   char *stageEndTime;
   char *linkTypes;
   char *stageDesc;
   char *instList;
   char *cpuList;
   time_t stageElapsed;
   char *pidList;
   int stageStatus;
   char *custInfoList
} info;
} DSSTAGEINFO;
```

Members

infoType is a key indicating the information to be returned and is one of the following:

This key...
Indicates this information...

DSJ_LINKLIST
Null-separated list of link names.

DSJ_STAGELASTERR
The last error message generated from any link in the stage.

DSJ_STAGENAME
Name of stage.

DSJ_STAGETYPE
The stage type name, for example, Transformer or BeforeJob.

DSJ_STAGEINROWNUM
The primary link's input row number.

DSJ_VARLIST
List of stage variable names.

DSJ_STAGESTARTTIME-STAMP
Date and time when stage started.

DSJ_STAGEENDTIME-STAMP
Date and time when stage finished.

DSJ_STAGEDESC
Stage description (from stage properties)

DSJ_STAGEINST
Null-separated list of instance ids (parallel jobs).

DSJ_STAGECPU
Null-separated list of CPU time in seconds

DSJ_LINKTYPES
Null-separated list of link types.

DSJ_STAGEELAPSED
Elapsed time in seconds.

DSJ_STAGEPID
Null-separated list of process ids.

DSJ_STAGESTATUS
Stage status.

DSJ_CUSTINFOLIST
Null-separated list of custinfo item names.

lastError is a data structure containing the error message for the last error (if any) reported from any link of the stage. It is returned when *infoType* is set to DSJ_STAGELASTERR.

typeName is the stage type name and is returned when *infoType* is set to DSJ_STAGETYPE.

inRowNum is the primary link's input row number and is returned when *infoType* is set to DSJ_STAGEINROWNUM.

linkList is a pointer to a buffer that contains a series of null-terminated strings, one for each link in the stage, ending with a second null character, as shown in the following example (<null> represents the terminating null character):

```
first<null>second<null><null>
```

DSVARINFO

Use the DSVARINFO structure to represent various information values about a link to or from an active stage within an InfoSphere DataStage job.

Syntax

```
typedef struct _DSVARINFO {
int infoType:   /
union {
   char *varValue;
   char *varDesc;
} info;
} DSVARINFO;
```

Members

infoType is a key indicating the type of information and is one of the following values:

This key...
Indicates this information...

DSJ_VARVALUE
The value of the specified variable.

DSJ_VARDESC
The description of the specified variable.

Error Codes

Use error codes to determine application problems.

The following table lists InfoSphere DataStage API error codes in alphabetic order:

Table 2. API Error Codes

Error Token	Code	Description
DSJE_ACCOUNTPATHFAILED	-140	Failed to get account path.
DSJE_ADDPROJECTBLOCKED	-134	Another user is adding a project.
DSJE_ADDPROJECTFAILED	-135	Failed to add project.
DSJEBADBOOLEANVALUE	-118	Invalid value given for a boolean environment variable.
DSJE_BADENVVAR	-116	Environment variable does not exist.
DSJE_BADENVVARNAME	-108	Invalid environment variable name.
DSJE_BADENVVARTYPE	-109	Invalid environment variable type.
DSJE_BADENVVARPROMPT	-110	No prompt supplied.
DSJE_BADHANDLE	-1	Invalid *JobHandle.*
DSJE_BADLINK	-9	*LinkName* does not refer to a known link for the stage in question.
DSJE_BADLISTVALUE	-120	Invalid value given for a list environment variable.
DSJE_BADNAME	-12	Invalid project name.
DSJE_BADNUMERICVALUE	-119	Invalid value given for an integer environment variable.
DSJE_BADPARAM	-3	*ParamName* is not a parameter name in the job.
DSJE_BADPROJECT	-1002	*ProjectName* is not a known InfoSphere DataStage project.
DSJE_BADPROJLOCATION	-130	Invalid path name supplied.

Table 2. API Error Codes (continued)

Error Token	Code	Description
DSJE_BADPROJNAME	-128	Invalid project name supplied.
DSJE_BADPROPERTY	-104	Unknown property name.
DSJE_BADPROPVALUE	-106	Invalid value for this property.
DSJE_BADSTAGE	-7	*StageName* does not refer to a known stage in the job.
DSJE_BADSTATE	-2	Job is not in the right state (compiled, not running).
DSJE_BADTIME	-13	Invalid *StartTime* or *EndTime* value.
DSJE_BADTYPE	-5	Information or event type was unrecognized.
DSJE_BAD_VERSION	-1008	The engine does not support this version of the InfoSphere DataStage API.
DSJE_BADVALUE	-4	Invalid *MaxNumber* value.
DSJE_CLEARSCHEDULEFAILED	-127	Failed to clear scheduled jobs for project.
DSJE_DECRYPTERR	-15	Failed to decrypt encrypted values.
DSJE_DELETEPROJECTBLOCKED	-138	Project locked by another user.
DSJE_DELPROJFAILED	-124	Failed to delete project definition.
DSJE_DELPROJFILESFAILED	-125	Failed to delete project files.
DSJE_DUPENVVARNAME	-115	Environment variable being added already exists.
DSJE_ENCODEFAILED	-123	Failed to encode an encrypted value.
DSJE_GETDEFAULTPATHFAILED	-129	Failed to determine default project directory.
DSJE_INCOMPATIBLE_SERVER	-1009	The engine version is incompatible with this version of the InfoSphere DataStage API.
DSJE_ISADMINFAILED	-101	Failed to determine whether user is an administrator.
DSJE_ISPARALLELLICENCED	-122	Failed to determine if parallel jobs are available.
DSJE_INVALIDPROJECTLOCATION	-131	Invalid path name supplied.
DSJE_JOBDELETED	-11	The job has been deleted.
DSJE_JOBLOCKED	-10	The job is locked by another process.
DSJE_LICENSEPROJECTFAILED	-136	Failed to license project.
DSJE_LISTSCEDULEFAILED	-126	Failed to get list of scheduled jobs for project.
DSJE_LOGTOFAILED	-141	Failed to log to UV account.
DSJE_NOACCESS	-16	Cannot get values, default values, or design default values for any job except the current job.
DSJE_NO_DATASTAGE	-1003	InfoSphere DataStage is not installed on the system.

Table 2. API Error Codes (continued)

Error Token	Code	Description
DSJE_NOERROR	0	No InfoSphere DataStage API error has occurred.
DSJE_NO_MEMORY	-1005	Failed to allocate dynamic memory.
DSJE_NOMORE	-1001	All events matching the filter criteria have been returned.
DSJE_NOTADMINUSER	-100	User is not an administrator.
DSJE_NOTAPROJECT	-139	Failed to log to project.
DSJE_NOT_AVAILABLE	-1007	The requested information was not found.
DSJE_NOTINSTAGE	-8	Internal engine error.
DSJE_NOTUSERDEFINED	-117	Environment variable is not user-defined and therefore cannot be deleted.
DSJE_OPENFAIL	-1004	The attempt to open the job failed - perhaps it has not been compiled.
DSJE_OPENFAILED	-132	Failed to open UV.ACCOUNT file.
DSJE_OSHVISIBLEFLAG	-107	Failed to get value for OSHVisible.
DSJE_PROPNOTSUPPORTED	-105	Unsupported property.
DSJE_PXNOTINSTALLED	-121	Environment variable is specific to parallel jobs which are not available.
DSJE_READENVVARDEFNS	-111	Failed to read environment variable definitions.
DSJE_READENVVARVALUES	-112	Failed to read environment variable values.
DSJE_READPROJPROPERTY	-102	Failed to read property.
DSJE_READUFAILED	-133	Failed to lock project create lock record.
DSJE_RELEASEFAILED	-137	Failed to release project create lock record.
DSJE_REPERROR	-99	General engine error.
DSJE_SERVER_ERROR	-1006	An unexpected or unknown error occurred in the engine.
DSJE_TIMEOUT	-14	The job appears not to have started after waiting for a reasonable length of time. (About 30 minutes.)
DSJE_UNKNOWN_JOBNAME	-201	The supplied job name cannot be found in the project.
DSJE_WRITEENVVARDEFNS	-113	Failed to write environment variable definitions.
DSJE_WRITEENVVARVALUES	-114	Failed to write environment variable values.
DSJE_WRITEPROJPROPERTY	-103	Property not supported.
DSJE_WRONGJOB	-6	Job for this *JobHandle* was not started from a call to **DSRunJob** by the current process.

The following table lists InfoSphere DataStage API error codes in numeric order:

Table 3. API error codes in numeric order

Code	Error Token	Description
0	DSJE_NOERROR	No InfoSphere DataStage API error has occurred.
-1	DSJE_BADHANDLE	Invalid *JobHandle.*
-2	DSJE_BADSTATE	Job is not in the right state (compiled, not running).
-3	DSJE_BADPARAM	*ParamName* is not a parameter name in the job.
-4	DSJE_BADVALUE	Invalid *MaxNumber* value.
-5	DSJE_BADTYPE	Information or event type was unrecognized.
-6	DSJE_WRONGJOB	Job for this *JobHandle* was not started from a call to **DSRunJob** by the current process.
-7	DSJE_BADSTAGE	*StageName* does not refer to a known stage in the job.
-8	DSJE_NOTINSTAGE	Internal engine error.
-9	DSJE_BADLINK	*LinkName* does not refer to a known link for the stage in question.
-10	DSJE_JOBLOCKED	The job is locked by another process.
-11	DSJE_JOBDELETED	The job has been deleted.
-12	DSJE_BADNAME	Invalid project name.
-13	DSJE_BADTIME	Invalid *StartTime* or *EndTime* value.
-14	DSJE_TIMEOUT	The job appears not to have started after waiting for a reasonable length of time. (About 30 minutes.)
-15	DSJE_DECRYPTERR	Failed to decrypt encrypted values.
-16	DSJE_NOACCESS	Cannot get values, default values, or design default values for any job except the current job.
-99	DSJE_REPERROR	General engine error.
-100	DSJE_NOTADMINUSER	User is not an administrator.
-101	DSJE_ISADMINFAILED	Failed to determine whether user is an administrator.
-102	DSJE_READPROJPROPERTY	Failed to read property.
-103	DSJE_WRITEPROJPROPERTY	Property not supported.
-104	DSJE_BADPROPERTY	Unknown property name.
-105	DSJE_PROPNOTSUPPORTED	Unsupported property.
-106	DSJE_BADPROPVALUE	Invalid value for this property.
-107	DSJE_OSHVISIBLEFLAG	Failed to get value for OSHVisible.
-108	DSJE_BADENVVARNAME	Invalid environment variable name.
-109	DSJE_BADENVVARTYPE	Invalid environment variable type.
-110	DSJE_BADENVVARPROMPT	No prompt supplied.

Table 3. API error codes in numeric order (continued)

Code	Error Token	Description
-111	DSJE_READENVVARDEFNS	Failed to read environment variable definitions.
-112	DSJE_READENVVARVALUES	Failed to read environment variable values.
-113	DSJE_WRITEENVVARDEFNS	Failed to write environment variable definitions.
-114	DSJE_WRITEENVVARVALUES	Failed to write environment variable values.
-115	DSJE_DUPENVVARNAME	Environment variable being added already exists.
-116	DSJE_BADENVVAR	Environment variable does not exist.
-117	DSJE_NOTUSERDEFINED	Environment variable is not user-defined and therefore cannot be deleted.
-118	DSJE_BADBOOLEANVALUE	Invalid value given for a boolean environment variable.
-119	DSJE_BADNUMERICVALUE	Invalid value given for an integer environment variable.
-120	DSJE_BADLISTVALUE	Invalid value given for a list environment variable.
-121	DSJE_PXNOTINSTALLED	Environment variable is specific to parallel jobs which are not available.
-122	DSJE_ISPARALLELLICENCED	Failed to determine if parallel jobs are available.
-123	DSJE_ENCODEFAILED	Failed to encode an encrypted value.
-124	DSJE_DELPROJFAILED	Failed to delete project definition.
-125	DSJE_DELPROJFILESFAILED	Failed to delete project files.
-126	DSJE_LISTSCHEDULEFAILED	Failed to get list of scheduled jobs for project.
-127	DSJE_CLEARSCHEDULEFAILED	Failed to clear scheduled jobs for project.
-128	DSJE_BADPROJNAME	Invalid project name supplied.
-129	DSJE_GETDEFAULTPATHFAILED	Failed to determine default project directory.
-130	DSJE_BADPROJLOCATION	Invalid path name supplied.
-131	DSJE_INVALIDPROJECTLOCATION	Invalid path name supplied.
-132	DSJE_OPENFAILED	Failed to open UV.ACCOUNT file.
-133	DSJE_READUFAILED	Failed to lock project create lock record.
-134	DSJE_ADDPROJECTBLOCKED	Another user is adding a project.
-135	DSJE_ADDPROJECTFAILED	Failed to add project.
-136	DSJE_LICENSEPROJECTFAILED	Failed to license project.
-137	DSJE_RELEASEFAILED	Failed to release project create lock record.
-138	DSJE_DELETEPROJECTBLOCKED	Project locked by another user.

Table 3. API error codes in numeric order (continued)

Code	Error Token	Description
-139	DSJE_NOTAPROJECT	Failed to log to project.
-140	DSJE_ACCOUNTPATHFAILED	Failed to get account path.
-141	DSJE_LOGTOFAILED	Failed to log to UV account.
-201	DSJE_UNKNOWN_JOBNAME	The supplied job name cannot be found in the project.
-1001	DSJE_NOMORE	All events matching the filter criteria have been returned.
-1002	DSJE_BADPROJECT	*ProjectName* is not a known InfoSphere DataStage project.
-1003	DSJE_NO_DATASTAGE	InfoSphere DataStage is not installed on the system.
-1004	DSJE_OPENFAIL	The attempt to open the job failed - perhaps it has not been compiled.
-1005	DSJE_NO_MEMORY	Failed to allocate dynamic memory.
-1006	DSJE_SERVER_ERROR	An unexpected or unknown error occurred in the engine.
-1007	DSJE_NOT_AVAILABLE	The requested information was not found.
-1008	DSJE_BAD_VERSION	The engine does not support this version of the InfoSphere DataStage API.
-1009	DSJE_INCOMPATIBLE_SERVER	The engine version is incompatible with this version of the InfoSphere DataStage API.

The following table lists some common errors that might be returned from the lower-level communication tiers:

Table 4. API Communication Layer Error Codes

Error Number	Description
39121	The InfoSphere DataStage license has expired.
39134	The InfoSphere DataStage user limit has been reached.
80011	Incorrect system name or invalid user name or password provided.
80019	Password has expired.

InfoSphere DataStage BASIC Interface

Use InfoSphere DataStage BASIC functions to perform various tasks.

These functions can be used in a job control routine, which is defined as part of a job's properties and allows other jobs to be run and be controlled from the first job. Some of the functions can also be used for getting status information about the current job; these are useful in active stage expressions and before- and after-stage subroutines.

Table 5. BASIC Functions

To do this...	Use this...
Specify the job you want to control	DSAttachJob
Set parameters for the job you want to control	DSSetParam
Set limits for the job you want to control	DSSetJobLimit
Request that a job is run	DSRunJob
Wait for a called job to finish	DSWaitForJob
Get information from certain parallel stages.	DSGetCustInfo
Get information about the current project	DSGetProjectInfo
Get information about the controlled job or current job	DSGetJobInfo
Get information about a stage in the controlled job or current job	DSGetStageInfo
Get information about a link in a controlled job or current job	DSGetLinkInfo
Get information about a controlled job's parameters	DSGetParamInfo
Get the log event from the job log	DSGetLogEntry
Get the log event from the job log, including the message ID and the invocation ID	DSGetLogEntryFull
Get a list of log event IDs for a given run of a job invocation	DSGetLogEventIds
Get a number of log events on the specified subject from the job log	DSGetLogSummary
Get the newest log event, of a specified type, from the job log	DSGetNewestLogId
Log an event to the job log of a different job	DSLogEvent
Stop a controlled job	DSStopJob
Return a job handle previously obtained from **DSAttachJob**	DSDetachJob
Log a fatal error message in a job's log file and aborts the job.	DSLogFatal
Log an information message in a job's log file.	DSLogInfo
Put an info message in the job log of a job controlling current job.	DSLogToController
Log a warning message in a job's log file.	DSLogWarn
Generate a string describing the complete status of a valid attached job.	DSMakeJobReport
Insert arguments into the message template.	DSMakeMsg
Ensure a job is in the correct state to be run or validated.	DSPrepareJob
Interface to system send mail facility.	DSSendMail
Log a warning message to a job log file.	DSTransformError
Convert a job control status or error code into an explanatory text message.	DSTranslateCode

Table 5. BASIC Functions (continued)

To do this...	Use this...
Suspend a job until a named file either exists or does not exist.	DSWaitForFile
Checks if a BASIC routine is cataloged, either in VOC as a callable item, or in the catalog space.	DSCheckRoutine
Execute a DOS or engine command from a before/after subroutine.	DSExecute
Set a status message for a job to return as a termination message when it finishes	DSSetUserStatus
Specifies whether a job generates operational metadata as it runs. This overrides the default setting for the project.	DSSetGenerateOpMetaData

DSAttachJob

Use the DSAttachJob function to run a job in job control sequence. When you attach this function to a job, a handle is returned that is used for addressing the job. There can only be one handle open for a particular job at any one time.

Syntax

```
JobHandle = DSAttachJob (JobName, ErrorMode)
```

JobHandle is the name of a variable to hold the return value which is subsequently used by any other function or routine when referring to the job. Do not assume that this value is an integer.

JobName is a string giving the name of the job to be attached to.

ErrorMode is a value specifying how other routines using the handle should report errors. It is one of:

- DSJ.ERRFATAL Log a fatal message and abort the controlling job (default).
- DSJ.ERRWARNING Log a warning message but carry on.
- DSJ.ERRNONE No message logged - caller takes full responsibility (failure of DSAttachJob itself will be logged, however).

Remarks

A job cannot attach to itself.

The *JobName* parameter can specify either an exact version of the job in the form *job*%Rel*n.n.n*, or the latest version of the job in the form *job*. If a controlling job is itself released, you will get the latest released version of *job*. If the controlling job is a development version, you will get the latest development version of *job*.

Example

This is an example of attaching to Release 11 of the job Qsales:

```
Qsales_handle = DSAttachJob ("Qsales%Rel1",
→ DSJ.ERRWARN)
```

DSCheckRoutine

Use the DSCheckRoutine function to see if a BASIC routine is catalogued, either in the VOC as a callable item, or in the catalog space.

Syntax

```
Found = DSCheckRoutine(RoutineName)
```

RoutineName is the name of BASIC routine to check.

Found Boolean. @False if *RoutineName* not findable, else @True.

Example

```
rtn$ok = DSCheckRoutine("DSU.DSSendMail")
If(NOT(rtn$ok)) Then
   * error handling here
End.
```

DSDetachJob

Use the DSDetachJob function to get back a *JobHandle* parameter acquired by the DSAttachJob function if no further control of a job is required (allowing another job to become its controller). It is not necessary to call this function because attached jobs always detach automatically when the controlling job finishes.

Syntax

```
ErrCode = DSDetachJob (JobHandle)
```

JobHandle is the handle for the job as derived from DSAttachJob.

ErrCode is 0 if DSStopJob is successful, otherwise it might be the following:

- DSJE.BADHANDLE Invalid *JobHandle*.

The only possible error is an attempt to close DSJ.ME. Otherwise, the call always succeeds.

Example

The following command detaches the handle for the job qsales:

```
Deterr = DSDetachJob (qsales_handle)
```

DSExecute

Use the DSExecute function to run a DOS, UNIX, or engine command from a before-stage subroutine or an after-stage subroutine.

Syntax

```
Call DSExecute (ShellType, Command, Output, SystemReturnCode)
```

ShellType (input) specifies the type of command that you want to execute and is NT, UNIX, or UV (for engine).

Command (input) is the command to execute. *Command* should not prompt for input when it is executed.

Output (output) is any output from the command. Each line of output is separated by a field mark, @FM. Output is added to the job log file as an information message.

SystemReturnCode (output) is a code indicating the success of the command. A value of 0 means the command executed successfully. A value of 1 (for a DOS or UNIX command) indicates that the command was not found. Any other value is a specific exit code from the command.

Remarks

Do not use DSExecute from a transform; the overhead of running a command for each row processed by a stage will degrade performance of the job.

DSGetCustInfo

Use the DSGetCustInfo function to obtain information reported at the end of the execution of certain parallel stages. At design time, specify the information collected and available to be interrogated. For example, you can specify transformer stage information in the Triggers tab of the Transformer stage Properties dialog box.

Syntax

```
Result = DSGetCustInfo (JobHandle, StageName, CustInfoName, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it might be DSJ.ME to refer to the current job.

StageName is the name of the stage to be interrogated. It might also be DSJ.ME to refer to the current stage if necessary.

CustInfoName is the name of the variable to be interrogated.

InfoType specifies the information required and can be one of:

DSJ.CUSTINFOVALUE

DSJ.CUSTINFODESC

Result depends on the specified *InfoType*, as follows:

- DSJ.CUSTINFOVALUE String - the value of the specified custinfo item.
- DSJ.CUSTINFODESC String - description of the specified custinfo item.

Result might also return an error condition as follows:

- DSJE.BADHANDLE *JobHandle* was invalid.
- DSJE.BADTYPE *InfoType* was unrecognized.
- DSJE.NOTINSTAGE *StageName* was DSJ.ME and the caller is not running within a stage.
- DSJE.BADSTAGE *StageName* does not refer to a known stage in the job.
- DSJE.BADCUSTINFO *CustInfoName* does not refer to a known custinfo item.

DSGetJobInfo

Use the DSGetJobInfo function to obtain information about a job. You can use this information generally as well as for job control. The DSGetJobInfo function can refer to the current job or a controlled job, depending on the value of the *JobHandle* variable.

Syntax

```
Result = DSGetJobInfo (JobHandle, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it might be DSJ.ME to refer to the current job.

InfoType specifies the information required and can be one of:

DSJ.JOBSTATUS

DSJ.JOBNAME

DSJ.JOBCONTROLLER

DSJ.JOBSTARTTIMESTAMP

DSJ.JOBWAVENO

DSJ.PARAMLIST

DSJ.STAGELIST

DSJ.USERSTATUS

DSJ.JOBCONTROL

DSJ.JOBPID

DSJ.JPBLASTTIMESTAMP

DSJ.JOBINVOCATIONS

DSJ.JOBINTERIMSTATUS

DSJ.JOBINVOCATIONID

DSJ.JOBDESC

DSJ.JOBFULLDESC

DSJ.STAGELIST2

DSJ.JOBELAPSED

DSJ.JOBEOTCOUNT

DSJ.JOBEOTTIMESTAMP

DSJ.JOBRTISERVICE

DSJ.JOBMULTIINVOKABLE

DSJ.JOBFULLSTAGELIST

Result depends on the specified *InfoType*, as follows:

- DSJ.JOBSTATUS *Integer*. Current status of job overall. Possible statuses that can be returned are currently divided into two categories:

  Firstly, a job that is in progress is identified by:

  DSJS.RESET Job finished a reset run.

  DSJS.RUNFAILED Job finished a normal run with a fatal error.

  DSJS.RUNNING Job running - this is the only status that means the job is actually running.

  Secondly, jobs that are not running might have the following statuses:

  DSJS.RUNOK Job finished a normal run with no warnings.

  DSJS.RUNWARN Job finished a normal run with warnings.

  DSJS.STOPPED Job was stopped by operator intervention (can't tell run type).

  DSJS.VALFAILED Job failed a validation run.

  DSJS.VALOK Job finished a validation run with no warnings.

  DSJS.VALWARN Job finished a validation run with warnings.
- DSJ.JOBNAME *String*. Actual name of the job referenced by the job handle.
- DSJ.JOBCONTROLLER *String*. Name of the job controlling the job referenced by the job handle. Note that this might be several job names separated by periods if the job is controlled by a job which is itself controlled.
- DSJ.JOBSTARTTIMESTAMP *String*. Date and time when the job started on the engine in the form *YYYY-MM-DD hh:nn:ss*.
- DSJ.JOBWAVENO *Integer*. Wave number of last or current run.
- DSJ.PARAMLIST. Returns a comma-separated list of parameter names.
- DSJ.STAGELIST. Returns a comma-separated list of active stage names.
- DSJ.USERSTATUS *String*. Whatever the job's last call of DSSetUserStatus last recorded, else the empty string.
- DSJ.JOBCONTROL *Integer*. Current job control status, that is, whether a stop request has been issued for the job.
- DSJ. JOBPID *Integer*. Job process id.
- DSJ.JOBLASTTIMESTAMP *String*. Date and time when the job last finished a run on the engine in the form *YYYY-MM-DD HH:NN:SS*.
- DSJ.JOBINVOCATIONS. Returns a comma-separated list of Invocation IDs.
- DSJ.JOBINTERIMSTATUS. Returns the status of a job after it has run all stages and controlled jobs, but before it has attempted to run an after-job subroutine. (Designed to be used by an after-job subroutine to get the status of the current job).
- DSJ.JOBINVOCATIONID. Returns the invocation ID of the specified job (used in the DSJobInvocationId macro in a job design to access the invocation ID by which the job is invoked).
- DSJ.STAGELIST2. Returns a comma-separated list of passive stage names.
- DSJ.JOBELAPSED *String*. The elapsed time of the job in seconds.
- DSJ.JOBDESC *string*. The Job Description specified in the Job Properties dialog box.
- DSJ.JOBFULLDESSC *string*. The Full Description specified in the Job Properties dialog box.

- DSJ.JOBRTISERVICE *integer*. Set to true if this is a Web service job.
- DSJ.JOBMULTIINVOKABLE *integer*. Set to true if this job supports multiple invocations
- DSJ.JOBEOTCOUNT *integer*. Count of EndOfTransmission blocks processed by this job so far.
- DSJ.JOBEOTTIMESTAMP *timestamp*. Date/time of the last EndOfTransmission block processed by this job.
- DSJ.FULLSTAGELIST. Returns a comma-separated list of all stage names.

Result might also return error conditions as follows:

DSJE.BADHANDLE *JobHandle* was invalid.

DSJE.BADTYPE *InfoType* was unrecognized.

Remarks

When referring to a controlled job, DSGetJobInfo can be used either before or after a DSRunJob has been issued. Any status returned following a successful call to DSRunJob is guaranteed to relate to that run of the job.

Examples

The following command requests the job status of the job qsales:

```
q_status = DSGetJobInfo(qsales_handle, DSJ.JOBSTATUS)
```

The following command requests the actual name of the current job:

```
whatname = DSGetJobInfo (DSJ.ME, DSJ.JOBNAME)
```

DSGetLinkInfo

Use the DSGetLinkInfo function to obtain information about a link on an active stage. You can use this information generally as well as for job control. The DSGetLinkInfo function might reference either a controlled job or the current job, depending on the value of the *JobHandle* variable.

Syntax

```
Result = DSGetLinkInfo (JobHandle, StageName, LinkName, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it can be DSJ.ME to refer to the current job.

StageName is the name of the active stage to be interrogated. might also be DSJ.ME to refer to the current stage if necessary.

LinkName is the name of a link (input or output) attached to the stage. might also be DSJ.ME to refer to current link (for example, when used in a Transformer expression or transform function called from link code).

InfoType specifies the information required and can be one of:

DSJ.LINKLASTERR

DSJ.LINKNAME

DSJ.LINKROWCOUNT

DSJ.LINKSQLSTATE

DSJ.LINKDBMSCODE

DSJ.LINKDESC

DSJ.LINKSTAGE

DSJ.INSTROWCOUNT

DSJ.LINKEOTROWCOUNT

Result depends on the specified *InfoType*, as follows:

- DSJ.LINKLASTERR String - last error message (if any) reported from the link in question.
- DSJ.LINKNAME String - returns the name of the link, most useful when used with *JobHandle* = DSJ.ME and *StageName* = DSJ.ME and *LinkName* = DSJ.ME to discover your own name.
- DSJ.LINKROWCOUNT Integer - number of rows that have passed down a link so far.
- DSJ.LINKSQLSTATE - the SQL state for the last error occurring on this link.
- DSJ.LINKDBMSCODE - the DBMS code for the last error occurring on this link.
- DSJ.LINKDESC - description of the link.
- DSJ.LINKSTAGE - name of the stage at the other end of the link.
- DSJ.INSTROWCOUNT - comma-separated list of row counts, one per instance (parallel jobs)
- DSJ.LINKEOTROWCOUNT - row count since last EndOfTransmission block.

Result might also return error conditions as follows:

- DSJE.BADHANDLE *JobHandle* was invalid.
- DSJE.BADTYPE *InfoType* was unrecognized.
- DSJE.BADSTAGE *StageName* does not refer to a known stage in the job.
- DSJE.NOTINSTAGE *StageName* was DSJ.ME and the caller is not running within a stage.
- DSJE.BADLINK *LinkName* does not refer to a known link for the stage in question.

Remarks

When referring to a controlled job, DSGetLinkInfo can be used either before or after a DSRunJob has been issued. Any status returned following a successful call to DSRunJob is guaranteed to relate to that run of the job.

Example

The following command requests the number of rows that have passed down the order_feed link in the loader stage of the job qsales:

```
link_status = DSGetLinkInfo(qsales_handle, "loader",
→ "order_feed", DSJ.LINKROWCOUNT)
```

DSGetLogEntry

Use the DSGetLogEntry function to read the full event details of a specific log event with the name *EventId*.

Syntax

```
EventDetail = DSGetLogEntry (JobHandle, EventId)
```

JobHandle is the handle for the job as derived from DSAttachJob.

EventId is an integer that identifies the specific log event for which details are required. This is obtained using the DSGetNewestLogId function.

EventDetail is a string containing substrings separated by \. The substrings are as follows:

Substring1 Timestamp in form *YYYY-MM-DD HH:NN:SS*

Substring2 User information

Substring3 EventType - see DSGetNewestLogId

Substring4 Event message

If an error occurs, the error is reported by one of the following negative integer result codes:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADVALUE Error accessing *EventId*.

Example

The following commands first get the EventID for the required log event and then reads full event details of the log event identified by LatestLogid into the string LatestEventString:

```
latestlogid =
→ DSGetNewestLogId(qsales_handle,DSJ.LOGANY)
LatestEventString =
→ DSGetLogEntry(qsales_handle,latestlogid)
```

DSGetLogEntryFull

Use the DSGetLogEntryFull function to read the full event details for an event, including the message ID and the invocation ID.

Syntax

```
EventDetail = DSGetLogEntryFull (JobHandle, EventId)
```

JobHandle is the handle for the job as derived from DSAttachJob.

EventId is an integer that identifies the specific log event for which details are required. This is obtained using the DSGetNewestLogId function.

EventDetail is a string containing substrings separated by \. The substrings are as follows:

Substring1 Timestamp in form *YYYY-MM-DD HH:NN:SS*

Substring2 User information

Substring3 EventType - see DSGetNewestLogId

Substring4 Event message

Substring5 Message ID

Substring6 Invocation ID

If an error occurs, the error is reported by one of the following negative integer result codes:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADVALUE Error accessing *EventId*.

Example

The following commands first get the EventID for the required log event and then reads full event details of the log event identified by LatestLogid into the string LatestEventString:

```
latestlogid =
→ DSGetNewestLogId(qsales_handle,DSJ.LOGANY)
LatestEventString =
→ DSGetLogEntryFull(qsales_handle,latestlogid)
```

DSGetLogEventIds

Use the DSGetLogEventIds function to return a list of log event IDs for a run of a job invocation.

Syntax

```
IdList = DSGetLogEventIds (JobHandle, RunNumber, EventTypeFilter)
```

JobHandle is the handle for the job as derived from DSAttachJob.

RunNumber identifies the job invocation run for which event IDs are returned. Usually a zero value requests IDs for the most recent run of the job invocation. To retrieve details for earlier runs, supply negative values, such as -1 for details about the run before the most recent, -2 for details about the run before that, and so forth. Where explicit run numbers are known, you can retrieve details by supplying the run number as a positive value.

EventTypeFilter restricts the types of event log entry for which IDs are returned. By default, IDs for all log entries are returned. Include characters in the filter string to restrict entries as follows:

I Informational

W Warning

F Fatal

S Start or End events

B Batch or Control events

R Purge or reset events

J Reject events

IdList is returned as a list of positive integers that identify the required log events. In the case of an error, *IdList* can also be returned as a negative integer, in which case it contains one of these error codes:

DSJE.BADHANDLE
Invalid *JobHandle*.

DSJE.BADTYPE
Invalid *EventTypeFilter*.

DSJE.BADVALUE
Invalid *RunNumber*.

Remarks

To use this method, the program needs to have previously acquired a job handle by calling DSAttachJob.

The run number for a job invocation is reset when the job is compiled, thus it is not possible to use this method to retrieve job event IDs for runs that occurred prior to the most recent job compilation.

DSGetLogSummary

Use the DSGetLogSummary function to return a list of short log event details. The details that are returned are determined by the settings of certain filters. Use care with the settings of the filters, otherwise a large amount of information can be returned.

Syntax

```
SummaryArray = DSGetLogSummary (JobHandle, EventType, StartTime, EndTime, MaxNumber)
```

JobHandle is the handle for the job as derived from *DSAttachJob*.

EventType is the type of event logged and is one of:

- DSJ.LOGINFO Information message
- DSJ.LOGWARNING Warning message
- DSJ.LOGFATAL Fatal error
- DSJ.LOGREJECT Reject link was active
- DSJ.LOGSTARTED Job started
- DSJ.LOGRESET Log was reset
- DSJ.LOGANY Any category (the default)

StartTime is a string in the form *YYYY-MM-DD HH:NN:SS* or *YYYY-MM-DD*.

EndTime is a string in the form *YYYY-MM-DD HH:NN:SS* or *YYYY-MM-DD*.

MaxNumber is an integer that restricts the number of events to return. 0 means no restriction. Use this setting with caution.

SummaryArray is a dynamic array of fields separated by @FM. Each field comprises a number of substrings separated by \, where each field represents a separate event, with the substrings as follows:

Substring1 *EventId* as per DSGetLogEntry

Substring2 Timestamp in form *YYYY-MM-DD HH:NN:SS*

Substring3 *EventType* - see DSGetNewestLogId

Substring4 - *n* Event message

If an error occurs, the error is reported by one of the following negative integer result codes:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADTYPE Invalid *EventType*.
- DSJE.BADTIME Invalid *StartTime* or *EndTime*.
- DSJE.BADVALUE Invalid *MaxNumber*.

Example

The following command produces an array of reject link active events recorded for the qsales job between 18th August 1998, and 18th September 1998, up to a maximum of MAXREJ entries:

```
RejEntries = DSGetLogSummary (qsales_handle,
→ DSJ.LOGREJECT, "1998-08-18 00:00:00", "1998-09-18
→ 00:00:00", MAXREJ)
```

DSGetNewestLogId

Use the DSGetNewestLogId function to get the ID of the most recent log event in a category.

Syntax

```
EventId = DSGetNewestLogId (JobHandle, EventType)
```

JobHandle is the handle for the job as derived from DSAttachJob.

EventType is the type of event logged and is one of:

- DSJ.LOGINFO Information message
- DSJ.LOGWARNING Warning message
- DSJ.LOGFATAL Fatal error
- DSJ.LOGREJECT Reject link was active
- DSJ.LOGSTARTED Job started
- DSJ.LOGRESET Log was reset
- DSJ.LOGANY Any category (the default)

EventId is a positive integer that identifies the specific log event. In the case of an error, *EventId* can also be returned as a negative integer, in which case it contains an error code as follows:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADTYPE Invalid *EventType*.

Example

The following command obtains an ID for the most recent warning message in the log for the qsales job:

```
Warnid = DSGetNewestLogId (qsales_handle,
→ DSJ.LOGWARNING)
```

DSGetParamInfo

Use the DSGetParamInfo function to obtain information about a parameter. You can use this information for job control. The DSGetParamInfo function might reference either a controlled job or the current job, depending on the value of the *JobHandle* variable.

Syntax

```
Result = DSGetParamInfo (JobHandle, ParamName, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it might be DSJ.ME to refer to the current job.

ParamName is the name of the parameter to be interrogated.

InfoType specifies the information required and might be one of:

DSJ.PARAMDEFAULT

DSJ.PARAMHELPTEXT

DSJ.PARAMPROMPT

DSJ.PARAMTYPE

DSJ.PARAMVALUE

DSJ.PARAMDES.DEFAULT

DSJ.PARAMLISTVALUES

DSJ.PARAMDES.LISTVALUES

DSJ.PARAMPROMPT.AT.RUN

Result depends on the specified *InfoType*, as follows:

- DSJ.PARAMDEFAULT String - Current® default value for the parameter in question. See also DSJ.PARAMDES.DEFAULT.
- DSJ.PARAMHELPTEXT String - Help text (if any) for the parameter in question.
- DSJ.PARAMPROMPT String - Prompt (if any) for the parameter in question.
- DSJ.PARAMTYPE Integer - Describes the type of validation test that should be performed on any value being set for this parameter. Is one of:
  DSJ.PARAMTYPE.STRING
  DSJ.PARAMTYPE.ENCRYPTED
  DSJ.PARAMTYPE.INTEGER
  DSJ.PARAMTYPE.FLOAT (the parameter might contain periods and E)
  DSJ.PARAMTYPE.PATHNAME
  DSJ.PARAMTYPE.LIST (should be a string of Tab-separated strings)
  DSJ.PARAMTYPE.DATE (should be a string in form *YYYY-MM-DD*)
  DSJ.PARAMTYPE.TIME (should be a string in form *HH:MM*)
- DSJ.PARAMVALUE String - Current value of the parameter for the running job or the last job run if the job is finished.

- DSJ.PARAMDES.DEFAULT String - Original default value of the parameter - might differ from DSJ.PARAMDEFAULT if the latter has been changed by an administrator since the job was installed.
- DSJ.PARAMLISTVALUES String - Tab-separated list of allowed values for the parameter. See also DSJ.PARAMDES.LISTVALUES.
- DSJ.PARAMDES.LISTVALUES String - Original Tab-separated list of allowed values for the parameter - might differ from DSJ.PARAMLISTVALUES if the latter has been changed by an administrator since the job was installed.
- DSJ.PROMPT.AT.RUN String - 1 means the parameter is to be prompted for when the job is run; anything else means it is not (DSJ.PARAMDEFAULT String to be used directly).

Result might also return error conditions as follows:

- DSJE.BADHANDLE *JobHandle* was invalid.
- DSJE.BADPARAM *ParamName* is not a parameter name in the job.
- DSJE.BADTYPE *InfoType* was unrecognized.

Remarks

When referring to a controlled job, DSGetParamInfo can be used either before or after a DSRunJob has been issued. Any status returned following a successful call to DSRunJob is guaranteed to relate to that run of the job.

Example

The following command requests the default value of the quarter parameter for the qsales job:

```
Qs_quarter = DSGetparamInfo(qsales_handle, "quarter",
→ DSJ.PARAMDEFAULT)
```

DSGetProjectInfo

Use the DSGetProjectInfo function to obtain information about the current project.

Syntax

```
Result = DSGetProjectInfo (InfoType)
```

InfoType specifies the information required and can be one of:

DSJ.JOBLIST

DSJ.PROJECTNAME

DSJ.HOSTNAME

Result depends on the specified *InfoType*, as follows:

- DSJ.JOBLIST String - comma-separated list of names of all jobs known to the project (whether the jobs are currently attached or not).
- DSJ.PROJECTNAME String - name of the current project.
- DSJ.HOSTNAME String - the host name of the engine holding the current project.

Result might also return an error condition as follows:

- DSJE.BADTYPE *InfoType* was unrecognized.

DSGetStageInfo

Use the DSGetStageInfo function to obtain information about a stage. You can use this information for job control. The DSGetStageInfo function can refer to either the current job or a controlled job, depending on the value of the *JobHandle* variable.

Syntax

```
Result = DSGetStageInfo (JobHandle, StageName, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it might be DSJ.ME to refer to the current job.

StageName is the name of the stage to be interrogated. It might also be DSJ.ME to refer to the current stage if necessary.

InfoType specifies the information required and might be one of:

DSJ.LINKLIST

DSJ.STAGELASTERR

DSJ.STAGENAME

DSJ.STAGETYPE

DSJ.STAGEINROWNUM

DSJ.VARLIST

DSJ.STAGESTARTTIMESTAMP

DSJ.STAGEENDTIMESTAMP

DSJ.STAGEDESC

DSJ.STAGEINST

DSJ.STAGECPU

DSJ.LINKTYPES

DSJ.STAGEELAPSED

DSJ.STAGEPID

DSJ.STAGESTATUS

DSJ.STAGEEOTCOUNT

DSJ.STAGEEOTTIMESTAMP

DSJ.CUSTINFOLIST

DSJ.STAGEEOTSTART

Result depends on the specified *InfoType*, as follows:

- DSJ.LINKLIST - comma-separated list of link names in the stage.
- DSJ.STAGELASTERR String - last error message (if any) reported from any link of the stage in question.
- DSJ.STAGENAME String - most useful when used with *JobHandle* = DSJ.ME and *StageName* = DSJ.ME to discover your own name.
- DSJ.STAGETYPE String - the stage type name (for example, "Transformer", "BeforeJob").
- DSJ. STAGEINROWNUM Integer - the primary link's input row number.
- DSJ.VARLIST - comma-separated list of stage variable names.
- DSJ.STAGESTARTTIMESTAMP - date/time that stage started executing in the form *YYY-MM-DD HH:NN:SS.*
- DSJ.STAGEENDTIMESTAMP - date/time that stage finished executing in the form *YYY-MM-DD HH:NN:SS.*
- DSJ.STAGEDESC - stage description.
- DSJ.STAGEINST - comma-separated list of instance ids (parallel jobs).
- DSJ.STAGECPU - integer percentage of CPU used.
- DSJ.LINKTYPES - comma-separated list of link types.
- DSJ.STAGEELAPSED - elapsed time in seconds.
- DSJ.STAGEPID - comma-separated list of process ids.
- DSJ.STAGESTATUS - stage status.
- DSJ.STAGEEOTCOUNT - Count of EndOfTransmission blocks processed by this stage so far.
- DSJ.STAGEEOTTIMESTAMP - Data/time of last EndOfTransmission block received by this stage.
- DSJ.CUSTINFOLIST - custom information generated by stages (parallel jobs).
- DSJ.STAGEEOTSTART - row count at start of current EndOfTransmission block.

Result might also return error conditions as follows:

- DSJE.BADHANDLE *JobHandle* was invalid.
- DSJE.BADTYPE *InfoType* was unrecognized.
- DSJE.NOTINSTAGE *StageName* was DSJ.ME and the caller is not running within a stage.
- DSJE.BADSTAGE *StageName* does not refer to a known stage in the job.

Remarks

When referring to a controlled job, DSGetStageInfo can be used either before or after a DSRunJob has been issued. Any status returned following a successful call to DSRunJob is guaranteed to relate to that run of the job.

Example

The following command requests the last error message for the loader stage of the job qsales:

```
stage_status = DSGetStageInfo(qsales_handle, "loader",
→ DSJ.STAGELASTERR)
```

DSGetVarInfo

Use the DSGetVarInfo function to obtain information about variables that are used in transformer stages.

Syntax

```
Result = DSGetVarInfo (JobHandle, StageName, VarName, InfoType)
```

JobHandle is the handle for the job as derived from DSAttachJob, or it might be DSJ.ME to refer to the current job.

StageName is the name of the stage to be interrogated. It might also be DSJ.ME to refer to the current stage if necessary.

VarName is the name of the variable to be interrogated.

InfoType specifies the information required and can be one of:

DSJ.VARVALUE

DSJ.VARDESCRIPTION

Result depends on the specified *InfoType*, as follows:

- DSJ.VARVALUE String - the value of the specified variable.
- DSJ.VARDESCRIPTION String - description of the specified variable.

Result might also return an error condition as follows:

- DSJE.BADHANDLE *JobHandle* was invalid.
- DSJE.BADTYPE *InfoType* was not recognized.
- DSJE.NOTINSTAGE *StageName* was DSJ.ME and the caller is not running within a stage.
- DSJE.BADVAR *VarName* was not recognized.
- DSJE.BADSTAGE *StageName* does not refer to a known stage in the job.

DSLogEvent

Use the DSLogEvent function to log an event message to a job other than the current one. (Use the DSLogInfo, DSLogFatal, or DSLogWarn function to log an event message to the current job.)

Syntax

```
ErrCode = DSLogEvent (JobHandle, EventType, EventMsg)
```

JobHandle is the handle for the job as derived from DSAttachJob.

EventType is the type of event logged and is one of:

- DSJ.LOGINFO Information message
- DSJ.LOGWARNING Warning message

EventMsg is a string containing the event message.

ErrCode is 0 if there is no error. Otherwise it contains one of the following errors:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADTYPE Invalid *EventType* (particularly note that you cannot place a fatal message in another job's log).

Example

The following command, when included in the msales job, adds the message "monthly sales complete" to the log for the qsales job:

```
Logerror = DsLogEvent (qsales_handle, DSJ.LOGINFO,
→ "monthly sales complete")
```

DSLogFatal

Use the DSLogFatal function to log a fatal error message in a job's log file and terminate the job.

Syntax

```
Call DSLogFatal (Message, CallingProgName)
```

Message (input) is the warning message you want to log. *Message* is automatically prefixed with the name of the current stage and the calling before/after subroutine.

CallingProgName (input) is the name of the before/after subroutine that calls the DSLogFatal subroutine.

Remarks

DSLogFatal writes the fatal error message to the job log file and aborts the job. DSLogFatal never returns to the calling before/after subroutine, so it should be used with caution. If a job stops with a fatal error, it must be reset by using the Director client before it can be rerun.

In a before/after subroutine, it is better to log a warning message (using DSLogWarn) and exit with a nonzero error code, which allows InfoSphere DataStage to stop the job cleanly.

DSLogFatal should not be used in a transform. Use DSTransformError instead.

Example

```
Call DSLogFatal("Cannot open file", "MyRoutine")
```

DSLogInfo

Use the DSLogInfo function to log an information message in a job's log file.

Syntax

```
Call DSLogInfo (Message, CallingProgName)
```

Message (input) is the information message you want to log. *Message* is automatically prefixed with the name of the current stage and the calling program.

CallingProgName (input) is the name of the transform or before/after subroutine that calls the DSLogInfo subroutine.

Remarks

DSLogInfo writes the message text to the job log file as an information message and returns to the calling routine or transform. If DSLogInfo is called during the test phase for a newly created routine in the repository, the two arguments are displayed in the results window.

Unlimited information messages can be written to the job log file. However, if a lot of messages are produced, the job might run slowly and the Director client might take some time to display the job log file.

Example

```
Call DSLogInfo("Transforming: ":Arg1, "MyTransform")
```

DSLogToController

Use the DSLogToController function to put an informational message in the log file of the job controlling another job.

Syntax

```
Call DSLogToController(MsgString)
```

MsgString is the text to be logged. The log event is of type Information.

Remarks

If the current job is not under control, a silent exit is performed.

Example

```
Call DSLogToController("This is logged to parent")
```

DSLogWarn

Use the DSLogWarn function to log a warning message in a job's log file.

Syntax

```
Call DSLogWarn (Message, CallingProgName)
```

Message (input) is the warning message you want to log. *Message* is automatically prefixed with the name of the current stage and the calling before/after subroutine.

CallingProgName (input) is the name of the before/after subroutine that calls the DSLogWarn subroutine.

Remarks

DSLogWarn writes the message to the job log file as a warning and returns to the calling before/after subroutine. If the job has a warning limit defined for it, when the number of warnings reaches that limit, the call does not return and the job is aborted.

DSLogWarn should not be used in a transform. Use DSTransformError instead.

Example

```
If InputArg > 100 Then
   Call DSLogWarn("Input must be =< 100; received
      ":InputArg,"MyRoutine")
End Else
   * Carry on processing unless the job aborts
End
```

DSMakeJobReport

Use the DSMakeJobReport function to generate a report describing the complete status of a valid attached job.

Syntax

```
ReportText = DSMakeJobReport(JobHandle, ReportLevel, LineSeparator)
```

JobHandle is the string as returned from DSAttachJob.

ReportLevel specifies the type of report and is one of the following:

- 0 - basic report. Text string containing start/end time, time elapsed and status of job.
- 1 - stage/link detail. As basic report, but also contains information about individual stages and links within the job.
- 2 - text string containing full XML report.

By default the generated XML will not contain a <?xml-stylesheet?> processing instruction. If a stylesheet is required, specify a ReportLevel of 2 and append the name of the required stylesheet URL, that is, 2;styleSheetURL. This inserts a processing instruction into the generated XML of the form:

```
<?xml-stylesheet type=text/xsl" href="styleSheetURL"?>
```

LineSeparator is the string used to separate lines of the report. Special values recognized are:

- "CRLF" => CHAR(13):CHAR(10)
- "LF" => CHAR(10)
- "CR" => CHAR(13)

The default is CRLF if on Windows, else LF.

Remarks

If a bad job handle is given, or any other error is encountered, information is added to the ReportText.

Example

```
h$ = DSAttachJob("MyJob", DSJ.ERRNONE)
rpt$ = DSMakeJobReport(h$,0,"CRLF")
```

DSMakeMsg

Use the DSMakeMsg function to insert arguments into a message template. Optionally, you can use the function to look up a template ID in the standard InfoSphere DataStage message file, and use any returned message template instead of that given to the routine.

Syntax

```
FullText = DSMakeMsg(Template, ArgList)
```

FullText is the message with parameters substituted

Template is the message template, in which %1, %2 and so on are to be substituted with values from the equivalent position in *ArgList*. If the template string starts

with a number followed by "\", that is assumed to be part of a message id to be looked up in the InfoSphere DataStage message file.

Note: If an argument token is followed by "[E]", the value of that argument is assumed to be a job control error code, and an explanation of it will be inserted in place of "[E]". (See the DSTranslateCode function.)

ArgList is the dynamic array, one field per argument to be substituted.

Remarks

This routine is called from job control code created by the JobSequence Generator.

It will also perform local job parameter substitution in the message text. That is, if called from within a job, it looks for substrings such as "#xyz#" and replaces them with the value of the job parameter named "xyz".

Example

```
t$ = DSMakeMsg("Error calling DSAttachJob(%1)<L>%2",
→jb$:@FM:DSGetLastErrorMsg())
```

DSPrepareJob

Use the DSPrepareJob function to ensure that a compiled job is in the correct state to be run or validated.

Syntax

```
JobHandle = DSPrepareJob(JobHandle)
```

JobHandle is the handle, as returned from DSAttachJob(), of the job to be prepared.

JobHandle is either the original handle or a new one. If returned as 0, an error occurred and a message is logged.

Example

```
h$ = DSPrepareJob(h$)
```

DSRunJob

Use the DSRunJob function to start running a job. This call is asynchronous; the request is passed to the runtime engine, but you are not informed of its progress.

Syntax

```
ErrCode = DSRunJob (JobHandle, RunMode)
```

JobHandle is the handle for the job as derived from DSAttachJob.

RunMode is the name of the mode that the job is to be run in and is one of:

- DSJ.RUNNORMAL (Default) Standard job run.
- DSJ.RUNRESET Job is to be reset.
- DSJ.RUNVALIDATE Job is to be validated only.
- DSJ.RUNRESTART Restartable job sequence is to be restarted with the original job parameter values.

ErrCode is 0 if DSRunJob is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADSTATE Job is not in the right state (compiled, not running).
- DSJE.BADTYPE *RunMode* is not a known mode.

Remarks

If the controlling job is running in validate mode, then any calls of DSRunJob will act as if *RunMode* was DSJ.RUNVALIDATE, regardless of the actual setting.

A job in validate mode will run its JobControl routine (if any) rather than just check for its existence, as is the case for before/after routines. This allows you to examine the log of what jobs it started up in validate mode.

After a call of DSRunJob, the controlled job's handle is unloaded. If you require to run the same job again, you must use DSDetachJob and DSAttachJob to set a new handle. Note that you will also need to use DSWaitForJob, as you cannot attach to a job while it is running.

Example

The following command starts the job qsales in standard mode:

```
RunErr = DSRunJob(qsales_handle, DSJ.RUNNORMAL)
```

DSSendMail

Use the DSSendMail function as an interface to a sendmail program that is assumed to exist somewhere in the search path of the current user (on the engine tier host). The function hides the different call interfaces to various sendmail programs, and provides a simple interface for sending text.

Syntax

```
Reply = DSSendMail(Parameters)
```

Parameters is a set of name:value parameters, separated by either a mark character or "\n".

Currently recognized names (case-insensitive) are:

- "From" Mail address of sender, for example, Me@SomeWhere.com

  Can only be left blank if the local template file does not contain a "%from%" token.
- "To" Mail address of recipient, for example, You@ElseWhere.com

  Can only be left blank if the local template file does not contain a "%to%" token.
- "Subject" Something to put in the subject line of the message.

  Refers to the "%subject%" token. If left as "", a standard subject line will be created, along the lines of "From InfoSphere DataStage job: jobname"
- "Server" Name of host through which the mail should be sent.

  might be omitted on systems (such as Unix) where the SMTP host name can be and is set up externally, in which case the local template file presumably will not contain a "%server%" token.
- "Body" Message body.

Can be omitted. An empty message will be sent. If used, it must be the last parameter, to allow for getting multiple lines into the message, using "\n" for line breaks. Refers to the "%body%" token.

Note: The text of the body might contain the tokens "%report% or %fullreport% anywhere within it, which will cause a report on the current job status to be inserted at that point. A full report contains stage and link information as well as job status.

Reply. Possible replies are:

- DSJE.NOERROR (0) OK
- DSJE.NOPARAM Parameter name missing - field does not look like 'name:value'
- DSJE.NOTEMPLATE Cannot find template file
- DSJE.BADTEMPLATE Error in template file

Remarks

The routine looks for a local file, in the current project directory, with a well-known name. That is, a template to describe exactly how to run the local sendmail command.

Example

```
code = DSSendMail("From:me@here\nTo:You@there\nSubject:Hi ya\nBody:Line1\nLine2")
```

DSSetDisableJobHandler

Use the DSSetDisableJobHandler function to enable or disable job-level message handling.

Syntax

```
ErrCode = DSSetDisableJobHandler (JobHandle, value)
```

JobHandle is the handle for the job as derived from **DSAttachJob**.

value is TRUE to disable job-level message handling, or FALSE to enable job-level message handling.

ErrCode is 0 if DSSetDisableJobHandler is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *JobHandle.*
- DSJE.BADVALUE *value* is not appropriate for that parameter type.

Example

The following command disables job-level message handling for the qsales job:

```
GenErr = DSSetDisableJobHandler (qsales_handle, TRUE)
```

DSSetDisableProjectHandler

Use the DSSetDisableProjectHandler function to enable or disable project-level message handling.

Syntax

```
ErrCode = DSSetDisableProjectHandler (ProjectHandle, value)
```

ProjectHandle is the value returned from **DSOpenProject**.

value is TRUE to disable project-level message handling, or FALSE to enable project-level message handling.

ErrCode is 0 if DSSetDisableProjectHandler is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *ProjectHandle.*
- DSJE.BADVALUE *value* is not appropriate for that parameter type.

Example

The following command disables project-level message handling for the qsales project:

```
GenErr = DSSetDisableProjectHandler (qsales_handle, TRUE)
```

DSSetGenerateOpMetaData

Use the DSSetGenerateOpMetaData function to specify whether the job generates operational metadata or not. This function overrides the default setting for the project.

Syntax

```
ErrCode = DSSetGenerateOpMetaData (JobHandle, value)
```

JobHandle is the handle for the job as derived from DSAttachJob.

value is TRUE to generate operational metadata, FALSE to not generate operational metadata.

ErrCode is 0 if DSRunJob is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *JobHandle*.
- DSJE.BADTYPE *value* is wrong.

Example

The following command causes the job qsales to generate operational metadata whatever the project default specifies:

GenErr = DSSetGenerateOpMetaData(qsales_handle, TRUE)

DSSetJobLimit

Use the DSSetJobLimit function to override the row or warning limits from the controlling job. By default a controlled job inherits any row or warning limits from the controlling job.

Syntax

```
ErrCode = DSSetJobLimit (JobHandle, LimitType, LimitValue)
```

JobHandle is the handle for the job as derived from DSAttachJob.

LimitType is the name of the limit to be applied to the running job and is one of:

- DSJ.LIMITWARN Job to be stopped after *LimitValue* warning events.
- DSJ.LIMITROWS Stages to be limited to *LimitValue* rows.

LimitValue is an integer specifying the value to set the limit to. Set this to 0 to specify unlimited warnings.

ErrCode is 0 if DSSetJobLimit is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *JobHandle.*
- DSJE.BADSTATE Job is not in the right state (compiled, not running).
- DSJE.BADTYPE *LimitType* is not a known limiting condition.
- DSJE.BADVALUE *LimitValue* is not appropriate for the limiting condition type.

Example

The following command sets a limit of 10 warnings on the qsales job before it is stopped:

```
LimitErr = DSSetJobLimit(qsales_handle,
→ DSJ.LIMITWARN, 10)
```

DSSetJobQueue

Sets the workload management queue before running a job.

Syntax

```
DSSetJobQueue (ErrCode, JobHandle, QueueName)
```

ErrCode is set to one of the following values:

- DSJE.NOERROR if DSSetJobQueue was successful
- DSJE.BADHANDLE if the *JobHandle* was not valid

JobHandle is the handle for the job as derived from DSAttachJob.

QueueName is the name of workload management queue to submit the job to.

Example

The following command sets the workload management queue to `HighCPUClass`.

```
Call DSSetJobQueue(ErrCode, qsales_handle, "HighCPUClass")
```

DSSetParam

Use the DSSetParam function to specify job parameter values before you run a job. Any parameter that is not set is defaulted.

Syntax

```
ErrCode = DSSetParam (JobHandle, ParamName, ParamValue)
```

JobHandle is the handle for the job as derived from DSAttachJob.

ParamName is a string giving the name of the parameter.

ParamValue is a string giving the value for the parameter.

ErrCode is 0 if DSSetParam is successful, otherwise it is one of the following negative integers:

- DSJE.BADHANDLE Invalid *JobHandle.*
- DSJE.BADSTATE Job is not in the right state (compiled, not running).
- DSJE.BADPARAM *ParamName* is not a known parameter of the job.
- DSJE.BADVALUE *ParamValue* is not appropriate for that parameter type.

Example 1

The following commands set the quarter parameter to 1 and the startdate parameter to 1/1/97 for the qsales job:

```
paramerr = DSSetParam (qsales_handle, "quarter", "1")
paramerr = DSSetParam (qsales_handle, "startdate", "1997-01-01")
```

Example 2

In this example, the qsales job uses a parameter set, called **PS1**, which contains two parameters, called **P1** and **P2** that have default values of `P1def` and `P2def`. There is a value set, called **VSetA** defined for the parameter set in which **P1** and **P2** have values of `P1A` and `P2A`.

By default, the job uses the default values from the parameter set **PS1**. The parameter values used are `P1=P1def` and `P2=P2def`

The following command sets the values of the parameters from the value set **VSetA**. The parameter values used are `P1=P1A` and `P2=P2A`.

```
paramerr = DSSetParam (qsales_handle, "PS1", "VsetA")
```

The following commands set the values of the parameters from the value set **VSetA**, and then override the value of the parameter **P2** with the value `P2X`. The parameter values used are `P1=P1A` and `P2=P2X`:

```
paramerr = DSSetParam (qsales_handle, "PS1", "VsetA")
paramerr = DSSetParam (qsales_handle, "PS1.P2", "P2x")
```

DSSetUserStatus

Use the DSSetUserStatus routine to set a termination code for interrogation by another job. This routine applies only to the current job, and does not take a *JobHandle* parameter.

This routine can be used by any job in either a JobControl or After routine to set a termination code for interrogation by another job. In fact, the code might be set at any point in the job, and the last setting is the one that will be picked up at any time. So to be certain of getting the actual termination code for a job the caller should use DSWaitForJob and DSGetJobInfo first, checking for a successful finishing status.

This routine is defined as a subroutine not a function because there are no possible errors.

Syntax

```
Call DSSetUserStatus (UserStatus)
```

UserStatus String is any user-defined termination message. The string will be logged as part of a suitable "Control" event in the calling job's log, and stored for retrieval by DSGetJobInfo, overwriting any previous stored string.

This string should not be a negative integer, otherwise it might be indistinguishable from an internal error in DSGetJobInfo calls.

Example

The following command sets a termination code of "sales job done":

```
Call DSSetUserStatus("sales job done")
```

DSStopJob

Use the DSStopJob function to immediately send a stop request to the runtime engine. Use this function only after a DSRunJob is issued.

The call is asynchronous. If you need to know that the job has actually stopped, you must call DSWaitForJob or use the Sleep statement and poll for DSGetJobStatus. The stop request is sent regardless of the job's current status.

Syntax

```
ErrCode = DSStopJob (JobHandle)
```

JobHandle is the handle for the job as derived from DSAttachJob.

ErrCode is 0 if DSStopJob is successful, otherwise it might be the following:

- DSJE.BADHANDLE Invalid *JobHandle*.

Example

The following command requests that the qsales job is stopped:

```
stoperr = DSStopJob(qsales_handle)
```

DSTransformError

Use the DSTransformerError function to log a warning message to a job log file. This function is called from transforms only.

Syntax

```
Call DSTransformError (Message, TransformName)
```

Message (input) is the warning message you want to log. *Message* is automatically prefixed with the name of the current stage and the calling transform.

TransformName (input) is the name of the transform that calls the DSTransformError subroutine.

Remarks

DSTransformError writes the message (and other information) to the job log file as a warning and returns to the transform. If the job has a warning limit defined for it, when the number of warnings reaches that limit, the call does not return and the job is aborted.

In addition to the warning message, DSTransformError logs the values of all columns in the current rows for all input and output links connected to the current stage.

Example

```
Function MySqrt(Arg1)
If Arg1 < 0 Then
   Call DSTransformError("Negative value:"Arg1, "MySqrt")
   Return("0")    ;*transform produces 0 in this case
End
   Result = Sqrt(Arg1) ;* else return the square root
Return(Result)
```

DSTranslateCode

Use the DSTranslateCode function to convert a job control status or error code into an explanatory text message.

Syntax

```
Ans = DSTranslateCode(Code)
```

Code is:

- If Code > 0, it's assumed to be a job status.
- If Code < 0, it's assumed to be an error code.
- (0 should never be passed in, and will return "no error")

Ans is the message associated with the code.

Remarks

If *Code* is not recognized, then *Ans* will report it.

Example

```
code$ = DSGetLastErrorMsg()
ans$ = DSTranslateCode(code$)
```

DSWaitForFile

Use the DSWaitForFile function to suspend a job until a named file either exists or does not exist.

Syntax

```
Reply = DSWaitForFile(Parameters)
```

Parameters is the full path of file to wait on. No check is made as to whether this is a reasonable path (for example, whether all directories in the path exist). A path name starting with "-", indicates a flag to check the nonexistence of the path. It is not part of the path name.

Parameters might also end in the form " timeout:NNNN" (or "timeout=NNNN") This indicates a non-default time to wait before giving up. There are several possible formats, case-insensitive:

- nnn number of seconds to wait (from now)
- nnnS ditto
- nnnM number of minutes to wait (from now)
- nnnH number of hours to wait (from now)

- nn:nn:nn wait until this time in 24HH:NN:SS. If this or nn:nn time has passed, will wait till next day.

The default timeout is the same as "12H".

The format might optionally terminate "/nn", indicating a poll delay time in seconds. If omitted, a default poll time is used.

Reply might be:
- DSJE.NOERROR (0) OK - file now exists or does not exist, depending on flag.
- DSJE.BADTIME Unrecognized Timeout format
- DSJE.NOFILEPATH File path missing
- DSJE.TIMEOUT Waited too long

Examples

```
Reply = DSWaitForFile("C:\ftp\incoming.txt timeout:2H")
```

(wait 7200 seconds for file on C: to exist before it gives up.)

```
Reply = DSWaitForFile("-incoming.txt timeout=15:00")
```

(wait until 3 p.m. for file in local directory to NOT exist.)

```
Reply = DSWaitForFile("incoming.txt timeout:3600/60")
```

(wait 1 hour for a local file to exist, looking once a minute.)

DSWaitForJob

Use the DSWaitForJob function to have a job or jobs wait before processing continues on the next BASIC statement.

This function is only valid if the current job has issued a DSRunJob on the given *JobHandle*(s). If one of the jobs whose handles are in the list has finished, the DSWaitForJob function returns immediately. If none of the jobs has finished, the DSWaitForJob function returns as soon as one of the jobs finishes.

Syntax

```
ErrCode = DSWaitForJob (JobHandle)
```

JobHandle is the string returned from DSAttachJob. If commas are contained, *JobHandle* is a comma-delimited set of job handles, representing a list of jobs to be waited for.

ErrCode is 0 if no error, else possible error values (<0) are:
- DSJE.BADHANDLE Invalid JobHandle.
- DSJE.WRONGJOB Job for this JobHandle was not run from within this job.

ErrCode is >0 => handle of the job that finished from a multi-job wait.

Remarks

DSWaitForJob waits for either a single job or multiple jobs.

Example

To wait for the return of the qsales job:

```
WaitErr = DSWaitForJob(qsales_handle)
```

Job Status Macros

Use macros that are provided in the JOBCONTROL.H file to obtain information about the current job, and links and stages belonging to the current job.

The macros that are provided in the JOBCONTROL.H file provide the functionality of using the InfoSphere DataStage BASIC **DSGetProjectInfo**, **DSGetJobInfo**, **DSGetStageInfo**, and **DSGetLinkInfo** functions with the DSJ.ME token as the *JobHandle* and can be used in all active stages and before/after subroutines. The macros provide the functionality for all the possible *InfoType* arguments for the **DSGet...Info** functions.

The available macros are:

- **DSHostName**
- **DSProjectName**
- **DSJobStatus**
- **DSJobName**
- **DSJobController**
- **DSJobStartDate**
- **DSJobStartTime**
- **DSJobWaveNo**
- **DSJobInvocations**
- **DSJobInvocationId**
- **DSStageName**
- **DSStageLastErr**
- **DSStageType**
- **DSStageInRowNum**
- **DSStageVarList**
- **DSLinkRowCount**
- **DSLinkLastErr**
- **DSLinkName**

For example, to obtain the name of the current job:

```
MyName = DSJobName
```

To obtain the full current stage name:

```
MyName = DSJobName : "." : DSStageName
```

In addition, the following macros are provided to manipulate Transformer stage variables:

- **DSGetVar**(*VarName*) returns the current value of the named stage variable. If the current stage does not have a stage variable called *VarName*, then "" is returned and an error message is logged. If the named stage variable is defined but has not been initialized, the "" is returned and an error message is logged.

- **DSSetVar**(*VarName*, *VarValue*) sets the value of the named stage variable. If the current stage does not have a stage variable called *VarName*, then an error message is logged.

Chapter 3. Generating an XML report

You can generate an XML report giving information about a job by using the following methods:

- DSMakeJobReport API function (see "DSMakeJobReport" on page 65)
- DSMakeJobReport BASIC function (see "DSMakeJobReport" on page 115)
- dsjob command (see "Generating a report" on page 13)

InfoSphere DataStage provides the following files to assist in the handling of generated XML reports:

- **DSReportSchema.xsd**. An XML schema document that fully describes the structure of the XML job report documents.
- **DSReport-Monitor.xsl**. An example XSLT stylesheet that creates an HTML web page similar to the Director Monitor view from the XML report.
- **DSReport-Waterfall.xsl**. An example XSLT stylesheet that creates an HTML web page showing a waterfall report describing how data flowed between all the processes in the job from the XML report.

The files are all located in the InfoSphere DataStage client directory (\IBM\InformationServer\Clients\Classic).

You can embed a reference to a stylesheet when you create the report using any of the commands listed above. After the report is generated you can view it in an Internet browser.

Alternatively you can use an xslt processor such as saxon or msxsl to convert an already generated report. For example:

```
java - jar saxon.jar jobreport.xml DSReport-Monitor.xsl > jobmonitor.htm
```

would generate an HTML file called jobmonitor.htm from the report jobreport.xml, while:

```
maxsl jobreport.xml DSReport-Waterfall.xsl > jobwaterfall.htm
```

would generate an HTML file called jobwaterfall.htm from the report jobreport.xml.

Appendix A. Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible.

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most web browsers. Because the information center uses XHTML, you can set display preferences in your browser. This also allows you to use screen readers and other assistive technologies to access the documentation.

The documentation that is in the information center is also provided in PDF files, which are not fully accessible.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Appendix B. Reading command-line syntax

This documentation uses special characters to define the command-line syntax.

The following special characters define the command-line syntax:

[ ] Identifies an optional argument. Arguments that are not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous argument.

| Indicates mutually exclusive information. You can use the argument to the left of the separator or the argument to the right of the separator. You cannot use both arguments in a single use of the command.

{ } Delimits a set of mutually exclusive arguments when one of the arguments is required. If the arguments are optional, they are enclosed in brackets ([ ]).

Note:

- The maximum number of characters in an argument is 256.
- Enclose argument values that have embedded spaces with either single or double quotation marks.

For example:

wsetsrc*[-S server] [-l label] [-n name] source*

The *source* argument is the only required argument for the **wsetsrc** command. The brackets around the other arguments indicate that these arguments are optional.

wlsac *[-l | -f format] [key...] profile*

In this example, the -l and -f format arguments are mutually exclusive and optional. The *profile* argument is required. The *key* argument is optional. The ellipsis (...) that follows the *key* argument indicates that you can specify multiple key names.

wrb -import *{rule_pack | rule_set}...*

In this example, the rule_pack and rule_set arguments are mutually exclusive, but one of the arguments must be specified. Also, the ellipsis marks (...) indicate that you can specify multiple rule packs or rule sets.

Appendix C. Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 6. IBM resources

Resource	Description and location
IBM Support Portal	You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server
Software services	You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/
My IBM	You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/
Training and certification	You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at http://www.ibm.com/training
IBM representatives	You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/

Appendix D. Accessing the product documentation

Documentation is provided in a variety of formats: in the online IBM Knowledge Center, in an optional locally installed information center, and as PDF books. You can access the online or locally installed help directly from the product client interfaces.

IBM Knowledge Center is the best place to find the most up-to-date information for InfoSphere Information Server. IBM Knowledge Center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open IBM Knowledge Center from the installed product or from a web browser.

Accessing IBM Knowledge Center

There are various ways to access the online documentation:

- Click the **Help** link in the upper right of the client interface.
- Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  Note: The F1 key does not work in web clients.
- Type the address in a web browser, for example, when you are not logged in to the product.

  Enter the following address to access all versions of InfoSphere Information Server documentation:

  `http://www.ibm.com/support/knowledgecenter/SSZJPZ/`

  If you want to access a particular topic, specify the version number with the product identifier, the documentation plug-in name, and the topic path in the URL. For example, the URL for the 11.3 version of this topic is as follows. (The ⇒ symbol indicates a line continuation):

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/⇒
  com.ibm.swg.im.iis.common.doc/common/accessingiidoc.html
  ```

  Tip:

  The knowledge center has a short URL as well:

  `http://ibm.biz/knowctr`

  To specify a short URL to a specific product page, version, or topic, use a hash character (#) between the short URL and the product identifier. For example, the short URL to all the InfoSphere Information Server documentation is the following URL:

  `http://ibm.biz/knowctr#SSZJPZ/`

  And, the short URL to the topic above to create a slightly shorter URL is the following URL (The ⇒ symbol indicates a line continuation):

  ```
  http://ibm.biz/knowctr#SSZJPZ_11.3.0/com.ibm.swg.im.iis.common.doc/⇒
  common/accessingiidoc.html
  ```

Changing help links to refer to locally installed documentation

IBM Knowledge Center contains the most up-to-date version of the documentation. However, you can install a local version of the documentation as an information center and configure your help links to point to it. A local information center is useful if your enterprise does not provide access to the internet.

Use the installation instructions that come with the information center installation package to install it on the computer of your choice. After you install and start the information center, you can use the **iisAdmin** command on the services tier computer to change the documentation location that the product F1 and help links refer to. (The ⇒ symbol indicates a line continuation):

Windows

```
IS_install_path\ASBServer\bin\iisAdmin.bat -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

AIX® Linux

```
IS_install_path/ASBServer/bin/iisAdmin.sh -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

Where <host> is the name of the computer where the information center is installed and <port> is the port number for the information center. The default port number is 8888. For example, on a computer named server1.example.com that uses the default port, the URL value would be http://server1.example.com:8888/help/topic/.

Obtaining PDF and hardcopy documentation

- The PDF file books are available online and can be accessed from this support document: https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1.
- You can also order IBM publications in hardcopy format online or through your local IBM representative. To order publications online, go to the IBM Publications Center at http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.

Appendix E. Providing feedback on the product documentation

You can provide helpful feedback regarding IBM documentation.

Your feedback helps IBM to provide quality information. You can use any of the following methods to provide comments:

- To provide a comment about a topic in IBM Knowledge Center that is hosted on the IBM website, sign in and add a comment by clicking **Add Comment** button at the bottom of the topic. Comments submitted this way are viewable by the public.
- To send a comment about the topic in IBM Knowledge Center to IBM that is not viewable by anyone else, sign in and click the **Feedback** link at the bottom of IBM Knowledge Center.
- Send your comments by using the online readers' comment form at www.ibm.com/software/awdtools/rcf/.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the information (if applicable). If you are commenting on specific text, include the location of the text (for example, a title, a table number, or a page number).

Notices and trademarks

This information was developed for products and services offered in the U.S.A. This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session or persistent cookies. If a product or component is not listed, that product or component does not use cookies.

Table 7. Use of cookies by InfoSphere Information Server products and components

Product module	Component or feature	Type of cookie that is used	Collect this data	Purpose of data	Disabling the cookies
Any (part of InfoSphere Information Server installation)	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
Any (part of InfoSphere Information Server installation)	InfoSphere Metadata Asset Manager	• Session • Persistent	No personally identifiable information	• Session management • Authentication • Enhanced user usability • Single sign-on configuration	Cannot be disabled

Table 7. Use of cookies by InfoSphere Information Server products and components (continued)

Product module	Component or feature	Type of cookie that is used	Collect this data	Purpose of data	Disabling the cookies
InfoSphere DataStage	Big Data File stage	• Session • Persistent	• User name • Digital signature • Session ID	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere DataStage	XML stage	Session	Internal identifiers	• Session management • Authentication	Cannot be disabled
InfoSphere DataStage	IBM InfoSphere DataStage and QualityStage® Operations Console	Session	No personally identifiable information	• Session management • Authentication	Cannot be disabled
InfoSphere Data Click	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Data Quality Console		Session	No personally identifiable information	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere QualityStage Standardization Rules Designer	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Information Governance Catalog		• Session • Persistent	• User name • Internal identifiers • State of the tree	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere Information Analyzer	Data Rules stage in the InfoSphere DataStage and QualityStage Designer client	Session	Session ID	Session management	Cannot be disabled

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACS^{Link}, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS^{Link} licensee of the United States Postal Service.

Other company, product or service names may be trademarks or service marks of others.

Index

Special characters

A

B

C

D

E

F

I

Printed in USA

SC19-4285-00

Spine information:

IBM InfoSphere DataStage **Version 11 Release 3** **Programmer's Guide** IBM