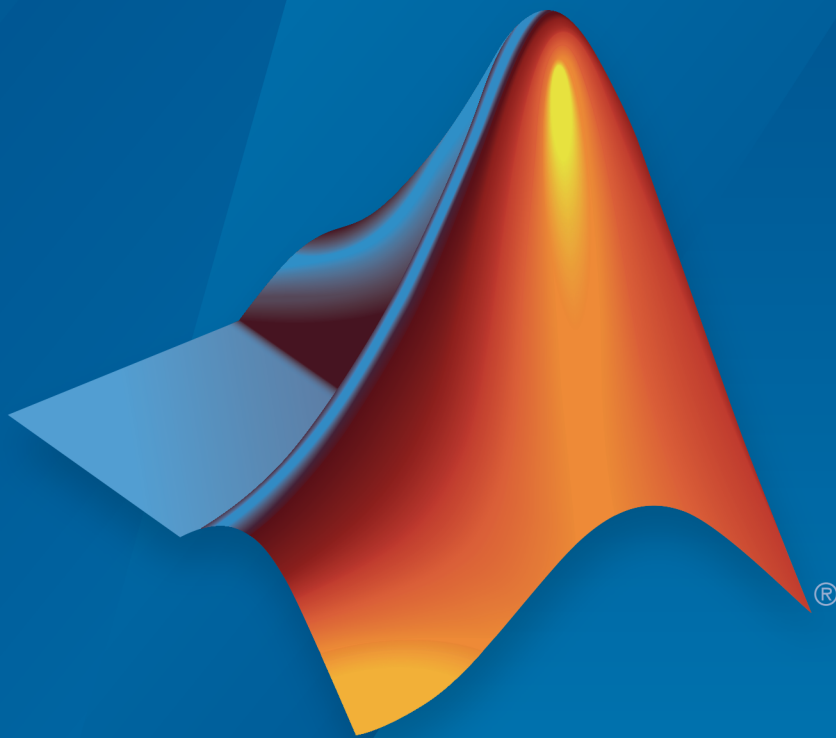


MATLAB[®] Compiler[™]
Hadoop[®] Integration Guide



MATLAB[®]

R2015a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

MATLAB[®] Compiler[™] Hadoop[®] Integration Guide

© COPYRIGHT 2014–2015 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

October 2014	Online only	New for Version 5.2 (R2014b)
March 2015	Online only	Revised for Version 6.0 (R2015a)

Deployable Archives

1

Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App	1-2
Create Deployable Archive to Run Against Hadoop from Command Line	1-6

Standalone Applications

2

Create Standalone Application to Run Against Hadoop from Command Line	2-2
--	-----

Hadoop Configuration

3

Hadoop Configuration	3-2
When Using Hadoop Standalone Mode	3-2
Hadoop Version Considerations	3-2
Hadoop Settings File	3-3

Functions — Alphabetical List

4

Apps

5

Deployable Archives

- “Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App” on page 1-2
- “Create Deployable Archive to Run Against Hadoop from Command Line” on page 1-6

Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App

This example shows how to create a deployable archive that calculates mean airline delays. It runs against Hadoop[®] using the Hadoop Compiler app, which is accessible from `deploytool`. The archive that you create contains all the MATLAB[®] based content associated with the component. The Hadoop Compiler app generates `mcc` commands that help you customize to your specification.

This example uses the `MaxMapReduceExample.m` example file and the airline dataset, `airlinesmall.csv`, both available at the `toolbox/matlab/demos` folder. The files in the folder are not part of the MATLAB Runtime. Move your example code to a new working folder for deployment.

Deployable archive that runs against Hadoop using Hadoop Compiler app is supported only on Linux[®].

Configure Files and Environment

Set environment variables and cluster properties for your Hadoop configuration. These properties are necessary for submitting jobs to your Hadoop cluster.

- Set up the environment variable, `HADOOP_HOME` to point at your Hadoop install folder. Modify the system path to include `$HADOOP_HOME/bin`.
- Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. The following example uses `/hd-shared/MCR/v84`.
- Copy the `airlinesmall.csv` into Hadoop Distributed File System (HDFS[™]) folder `/datasets/airlinemod`.
- Copy the map function `maxArrivalDelayMapper.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayMapper (data, info, intermKVStore)
partMax = max(data.ArrDelay);
add(intermKVStore, 'PartialMaxArrivalDelay', partMax);
```

For more information, see “Write a Map Function”.

- Copy the reduce function `maxArrivalDelayReducer.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
maxVal = -inf;
while hasNext(intermValIter)
    maxVal = max(getnext(intermValIter), maxVal);
end
add(outKVStore, 'MaxArrivalDelay', maxVal);
```

For more information, see “Write a Reduce Function”.

Create and Save Datastore

Create a datastore object from the `MaxMapReduceExample.m` and save the datastore to a `.mat` file.

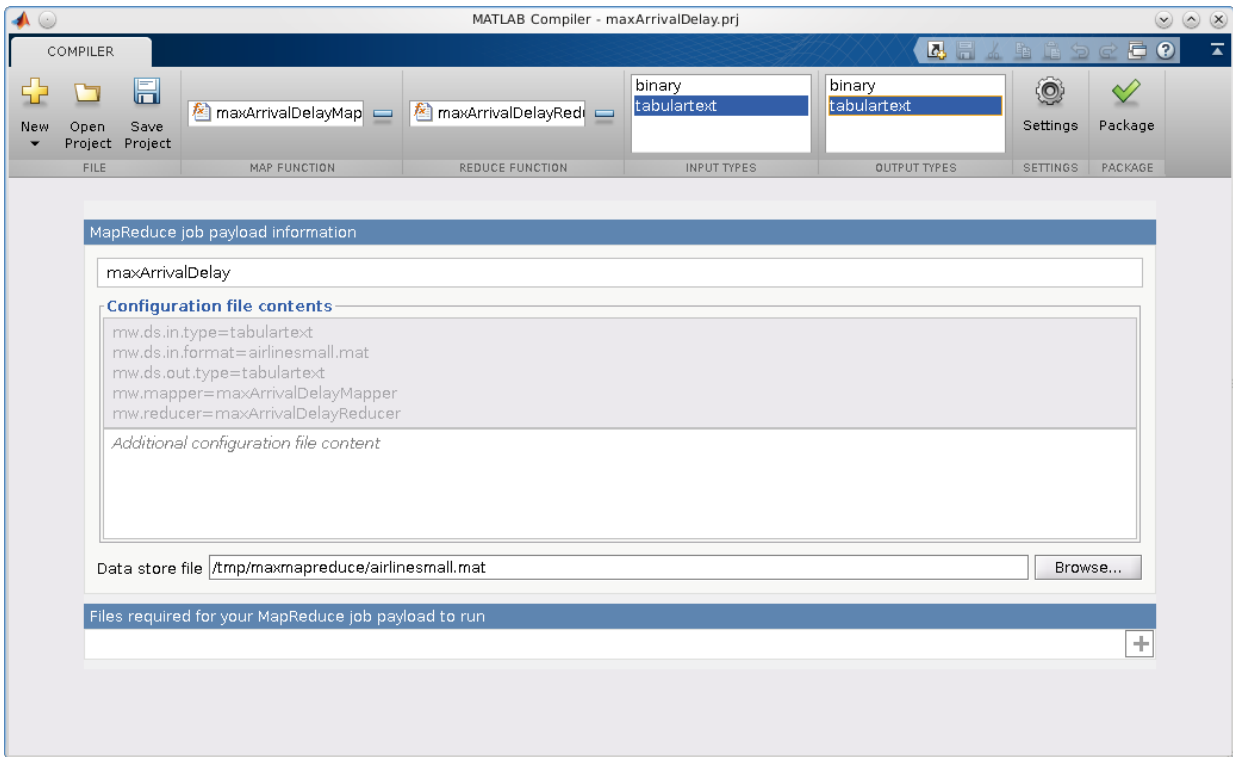
```
ds = datastore('airlinesmall.csv', 'TreatAsMissing', 'NA', ...
    'SelectedVariableNames', 'ArrDelay', 'ReadSize', 1000)
save('airlinesmall.mat', 'ds');
```

For more information, “Getting Started with Datastore”

Create a Deployable Archive Using Hadoop Compiler App

Launch the Hadoop Compiler app through the MATLAB command line or through the apps gallery. At the MATLAB command line type the following command:

```
hadoopCompiler
```



In the **Map Function** section of the toolstrip, click the plus button to add map file, which contains the map function. Browse and select one map function `maxArrivalDelayMapper.m`.

In the **Reduce Function** section of the toolstrip, click the plus button to add reduce file, which contains the reduce function. Browse and select one reduce function `maxArrivalDelayReducer.m`.

In the **Input Types** section, select `tabulartext` as input type. By default, the input type is `tabulartext`.

In the **Output Types** section, select `tabulartext` as output type. By default, the output type is `binary`.

Rename the application name to `maxArrivalDelay`.

In the **Data store file** field, click Browse and select the `airlinesmall.mat` file, which contains the saved datastore object.

Click **Package** to build a deployable archive.

The Hadoop Compiler app creates a log file `PackagingLog.txt` and two folders `for_redistribution` and `for_testing`. The `for_redistribution` folder contains readme file, shell script `run_maxarrivaldelay.sh`, and deployable archive `maxarrivaldelay.ctf`. The `for_testing` folder contains the same three files and a log file `mccExcludedfiles.log`.

Run Hadoop Job

At the MATLAB command prompt, run the deployable archive against Hadoop using the generated shell script. The arguments in the command are `MCRROOT`, Hadoop properties defined using `-D` flag, the data file, and the new results folder. The command to execute the script must be entered as a single line.

```
>> cd maxArrivalDelay/for_testing
>> !./run_maxarrivaldelay.sh /hd-shared/MCR/v84
-D mw.mcrroot=/hd-shared/MCR/v84 /datasets/airlinemod/airlinesmall.csv
myresults
```

View Results in MATLAB

Examine the results using the Hadoop command.

```
!./hadoop fs -cat myresults/*

'MaxArrivalDelay' [1014]
```

Other examples of map and reduce functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar deployable archives that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce”.

See Also

[datastore](#) | [deploytool](#) | [KeyValueDatastore](#) | [TabularTextDatastore](#)

Related Examples

- “Create Deployable Archive to Run Against Hadoop from Command Line” on page 1-6

Create Deployable Archive to Run Against Hadoop from Command Line

This example shows how to create a deployable archive with `mcc` command that calculates mean airline delays. The archive that you create contains all the MATLAB based content associated with the component. The `mcc` command creates a shell script to run the deployable archive against Hadoop. You can use shell script to customize the execution of the deployable archive within your particular Hadoop environment.

This example uses the `MaxMapReduceExample.m` example file and the airline dataset, `airlinesmall.csv`, both available at the `toolbox/matlab/demos` folder. The files in the folder are not part of the MATLAB Runtime. Move your example code to a new working folder for deployment.

Deployable archive that runs against Hadoop using `mcc` is supported only on Linux.

Configure Files and Environment

Set environment variables and cluster properties for your Hadoop configuration. These properties are necessary for submitting jobs to your Hadoop cluster.

- Set up the environment variable, `HADOOP_HOME` to point at your Hadoop install folder. Modify the system path to include `$HADOOP_HOME/bin`.
- Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. The following example uses `/hd-shared/MCR/v84`.
- Copy the `airlinesmall.csv` into Hadoop Distributed File System (HDFS) folder `/datasets/airlinemod`.
- Copy the map function `maxArrivalDelayMapper.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayMapper (data, info, intermKVStore)
partMax = max(data.ArrDelay);
add(intermKVStore, 'PartialMaxArrivalDelay', partMax);
```

For more information, see “Write a Map Function”.

- Copy the reduce function `maxArrivalDelayReducer.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
maxVal = -inf;
```

```

while hasNext(intermValIter)
    maxVal = max(getnext(intermValIter), maxVal);
end
add(outKVStore, 'MaxArrivalDelay', maxVal);

```

For more information, see “Write a Reduce Function”.

Create and Save Datastore

Create a datastore object from the `MaxMapReduceExample.m` and save the datastore to a `.mat` file.

```

ds = datastore('airlinesmall.csv', 'TreatAsMissing', 'NA', ...
    'SelectedVariableNames', 'ArrDelay', 'ReadSize', 1000)
save('airlinesmall.mat', 'ds');

```

For more information, “Getting Started with Datastore”

Create Hadoop Settings File

A Hadoop settings file specifies input type `tabulartext`, output type `binary`, the map function, the reduce function, and previously created `datastore`.

```

mw.ds.in.type=tabulartext
mw.ds.in.format=airlinesmall.mat
mw.ds.out.type=binary
mw.mapper=maxArrivalDelayMapper
mw.reducer=maxArrivalDelayReducer

```

For more information, see “Hadoop Settings File” on page 3-3.

Create a Deployable Archive Using `mcc`

Use the `mcc` command with the `-m` flag to create a deployable archive. The `-m` flag creates a standard executable that can be run from a command line. However, the `mcc` command cannot package the results in an installer. The command must be entered as a single line.

```

>> mcc -H -W 'hadoop:airlinesmall,CONFIG:MWHadoopSetting.txt'
    maxArrivalDelayMapper.m maxArrivalDelayReducer.m
    -a airlinesmall.mat

```

For more information, see `mcc`.

MATLAB Compiler™ creates a shell script `run_maxarrivaldelay.sh`, a deployable archive `airlinesmall.ctf`, and a log file `mccExcludedfiles.log`.

Run Hadoop Job

Deploy the archive as a Hadoop job by pointing the job to the csv files in the airline dataset. The arguments in the command are `MCRROOT`, Hadoop properties defined using `-D` flag, the data file, and the new results folder. The command must be entered as a single line.

```
>> !./run_airlinesmall.sh /hd-shared/MCR/v84  
-D mw.mcrroot=/hd-shared/MCR/v84 "/datasets/airline/*.csv"  
myresults
```

View Results in MATLAB

Visualize and plot the results.

```
ds= datastore('hdfs://hadoop01g1nxa64/user/username/myresults/part*',...  
'DatastoreType', 'keyvalue');  
airlinesmallResult = readall(ds)
```

Key	Value
'MaxArrivalDelay'	[1014]

Other examples of `map` and `reduce` functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar deployable archives that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce”.

See Also

`datastore` | `deploytool` | `KeyValueDatastore` | `mcc` | `TabularTextDatastore`

Related Examples

- “Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App” on page 1-2

Standalone Applications

Create Standalone Application to Run Against Hadoop from Command Line

This example shows how to modify a MATLAB example that calculates mean airline delays and creates a standalone application. The standalone application is a MATLAB program that runs against Hadoop using the `mcc` command. The `mapreducer` defines the environment for Hadoop.

This example uses the `MaxMapReduceExample.m` example file and the airline dataset, `airlinesmall.csv`, both available at the `toolbox/matlab/demos` folder. The files in the folder are not part of the MATLAB Runtime. Move your example code to a new working folder for deployment.

Standalone application that runs against Hadoop using `mcc` is supported only on Linux.

Configure Files and Environment

Set environment variables and cluster properties for your Hadoop configuration. These properties are necessary for submitting jobs to your Hadoop cluster.

- Set up the environment variable, `HADOOP_HOME` to point at your Hadoop install folder. Modify the system path to include `$HADOOP_HOME/bin`.

```
setenv ('HADOOP_HOME', '/share/hadoop/a1.2.1');
```

- Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. The following steps use `/hd-shared/MCR/v84`.
- Copy the `airlinesmall.csv` into Hadoop Distributed File System (HDFS) folder `/datasets/airlinemod`.
- Copy the map function `maxArrivalDelayMapper.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayMapper (data, info, intermKVStore)  
partMax = max(data.ArrDelay);  
add(intermKVStore, 'PartialMaxArrivalDelay', partMax);  
For more information, see "Write a Map Function".
```

- Copy the reduce function `maxArrivalDelayReducer.m` from `toolbox/matlab/demos` folder to the working folder.

```
function maxArrivalDelayReducer(intermKey, intermValIter, outKVStore)
```

```

maxVal = -inf;
while hasNext(intermediateValIter)
    maxVal = max(getnext(intermediateValIter), maxVal);
end
add(outKVStore, 'MaxArrivalDelay', maxVal);
For more information, see “Write a Reduce Function”.

```

Create Application using MapReduce

Create a `datastore` that points to the airline data in Hadoop Distributed File System (HDFS™).

```

ds = datastore('hdfs://hadoop01glnxa64/datasets/airlinemod/airlinesmall.csv', ...
    'TreatAsMissing', 'NA');
ds.SelectedVariableNames = {'Year', 'Month', 'DayofMonth', 'UniqueCarrier'};

```

If the files are located in HDFS, then the `datastore` should point to HDFS. For more information, see “Read from HDFS”.

Create a `mapreducer` object to set the properties of Hadoop in deployed mode. The `mapreducer` passes information about the execution environment to standalone applications that run against Hadoop. The `mapreducer` must point to the location of the MATLAB Runtime that is accessible from all the Hadoop worker nodes.

```

mr = mapreducer(matlab.mapreduce.DeployHadoopMapReducer('MCRRoot', ...
    '/hd-shared/hadoop-2.2.0/MCR/v84'));

```

For more information, see `matlab.mapreduce.DeployHadoopMapReducer`.

The new application `maxMapreduceapp.m` consists of a `datastore`, a `mapreducer` object that specifies the deployed environment variables, a `mapreduce` command, and a command to view the results of `mapreduce`:

```

ds = datastore('hdfs://hadoop01glnxa64/datasets/airlinemod/airlinesmall.csv', ...
    'TreatAsMissing', 'NA');
ds.SelectedVariableNames = {'Year', 'Month', 'DayofMonth', 'UniqueCarrier'};
mr = mapreducer(matlab.mapreduce.DeployHadoopMapReducer('MCRRoot', ...
    '/hd-shared/hadoop-2.2.0/MCR/v84'));
result = mapreduce(ds, @maxArrivalDelayMapper, @maxArrivalDelayReducer, mr, ...
    'OutputType', 'Binary', ...
    'OutputFolder', 'hdfs://hadoop01glnxa64/user/username/myresults' ...
    );
maxMapreduceappResult = readall(result)

```

Compile into Standalone Application

Use the `mcc` command with the `-m` flag to create a standalone application. The `-m` flag creates a standard executable that can be run from a command line. However, the `mcc` command cannot package the results in an installer.

```
mcc -m maxmapreduceapp.m
```

For more information, see `mcc`.

MATLAB Compiler creates `maxmapreduceapp.m`, shell script `run_maxarrivaldelay.sh`, and a log file `mccExcludedfiles.log`.

Run Standalone Application

Run the standalone application from MATLAB command prompt using the following command:

```
! ./maxmapreduce
```

Key	Value
'AA'	[92X1 double]
'AS'	[92X1 double]
'CO'	[92X1 double]
'DL'	[92X1 double]
'EA'	[92X1 double]

Results display in MATLAB.

Other examples of `map` and `reduce` functions are available at `toolbox/matlab/demos` folder. You can use other examples to prototype similar standalone applications that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce”.

See Also

`matlab.mapreduce.DeployHadoopMapReducer` | `datastore` | `KeyValueDatastore` | `mcc` | `TabularTextDatastore`

Related Examples

- “Package Standalone Application with Application Compiler App”

- “Pass Parallel Computing Toolbox Profile at Run Time”
- “Embed Parallel Computing Toolbox Profile”

Hadoop Configuration

- “Hadoop Configuration” on page 3-2
- “Hadoop Settings File” on page 3-3

Hadoop Configuration

In this section...
“When Using Hadoop Standalone Mode” on page 3-2
“Hadoop Version Considerations” on page 3-2

When Using Hadoop Standalone Mode

To execute a deployed MATLAB application or run a deployable archive as a Hadoop job in standalone mode, first set the appropriate environment variables in the Hadoop environment shell:

- Modify `HADOOP_CLASSPATH` according to your Hadoop version.
 - If you are working with Hadoop V1, use `mcr_root/toolbox/mlhadoop/jar/a1.2.1/mwmapreduce.jar`
 - If you are working with Hadoop V2, use `mcr_root/toolbox/mlhadoop/jar/a2.2.0/mwmapreduce.jar`
- Export `LD_LIBRARY_PATH` to include the following entries:
 - `mcr_root/runtime/glnxa64 :mcr_root/bin/glnxa64 mcr_root/sys/os/glnxa64 :mcr_root/sys/opengl/glnxa64`

where, `mcr_root` is the base of the install area for MATLAB Runtime

where, `mcr_root` is the base of the install area for MATLAB Runtime

Hadoop Version Considerations

- If you are working with Hadoop V1, improve the performance by setting `mapred.job.reuse.jvm.num.tasks` to `-1`.
- If you are working with Hadoop V2, the performance-improvement property is not supported.

Hadoop Settings File

In creating a deployable archive, you must create a Hadoop settings file that contains configuration details. If you are using `mcc`, create a text file. If you are using `deploytool`, the Hadoop Compiler app automatically creates the file for you when you select the map function, the reduce function, the input type, and the output type. You can view the contents of your settings file in the **Configuration file contents** section of the Hadoop Compiler app.

Parameter Type	Description	Default Value
<code>mw.mapper</code>	MATLAB map function name	Hadoop identity map function
<code>mw.reducer</code>	MATLAB reduce function name	Hadoop identity reduce function
<code>mw.ds.in.type</code>	MATLAB input type The input type is of two types, <code>tabulartext</code> and <code>binary</code> . The <code>tabulartext</code> input type is a formatted text file. The file is either a source file or result of the previous <code>mapreduce</code> job. The <code>binary</code> input type is a sequence file.	<code>tabulartext</code>
<code>mw.ds.in.forma</code>	This parameter is valid with <code>tabulartext</code> input type. This parameter specifies a <code>.mat</code> file that contains a <code>datastore</code> .	None
<code>mw.ds.in.reads</code>	This parameter is valid with <code>binary</code> input type. This parameter specifies a number that are number of rows for passing to the map function.	1
<code>mw.ds.out.type</code>	MATLAB output type The output type is of two types, <code>tabulartext</code> and <code>binary</code> . The <code>tabulartext</code> output type writes to a text file. The <code>binary</code> output type writes to a sequence file.	<code>binary</code>

This example shows a settings file with `tabulartext` input type:

```
mw.mapper=maxArrivalDelayMapper
mw.reducer=maxArrivalDelayReducer
mw.ds.in.type=tabulartext
mw.ds.in.format=airlinesmall.mat
mw.ds.out.type=tabulartext
```

This example shows a settings file with **binary** input type:

```
mw.mapper=maxArrivalDelayMapper
mw.reducer=maxArrivalDelayReducer
mw.ds.in.type=binary
mw.ds.in.readsize=1
mw.ds.out.type=tabulartext
```

Related Examples

- “Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App” on page 1-2
- “Create Deployable Archive to Run Against Hadoop from Command Line” on page 1-6

Functions — Alphabetical List

deploytool

Compile and package functions for external deployment

Syntax

```
deploytool
deploytool project_name
deploytool -build project_name
deploytool -package project_name
```

Description

`deploytool` opens a list of the compiler apps.

`deploytool project_name` opens the appropriate compiler app with the project preloaded.

`deploytool -build project_name` runs the appropriate compiler app to build the specified project. The installer is not generated.

`deploytool -package project_name` runs the appropriate compiler app to build and package the specified project. The installer is generated.

Examples

Create a New Compiler Project

Open the compiler to create a new project.

```
deploytool
```

Package an Application using an Existing Project

Open the compiler to build a new application using an existing project.


```
deploytool -package my_magic
```

Input Arguments

project_name — name of the project to be compiled
string

Specify the name of a previously saved project. The project must be on the current path.

mcc

Compile MATLAB functions for deployment

Syntax

```

mcc -e | -m [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g]
[-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w
option [:msg]] [-Y filename] mfilename
mcc -l [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g]
[-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w
option [:msg]] [-Y filename] mfilename...
mcc -c [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-g]
[-I directory...] [-K] [-M string] [-N] [-o filename] [-p path...] [-R option] [-v] [-w
option [:msg]] [-Y filename] mfilename...
mcc -W cpplib:library_name -T link:lib [-a filename...] [-B filename [:arg...]] [-C] [-d
outFolder] [-f filename] [-g] [-I directory...] [-K] [-M string] [-N] [-o filename] [-p
path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename...
mcc -W dotnet:assembly_name, [className], [framework_version], security,
remote_type -T link:lib [-a filename...] [-B filename [:arg...]] [-C] [-d outFolder] [-f
filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w
option [:msg]] [-Y filename] mfilename... [class{className: [mfilename...]}...]
mcc -W excel:addin_name, [className], [version] -T link:lib [-a filename...] [-b]
[-B filename [:arg...]] [-C] [-d outFolder] [-f filename] [-I directory...] [-K] [-M
string] [-N] [-p path...] [-R option] [-u] [-v] [-w option [:msg]] [-Y filename]
mfilename...
mcc -W 'java:packageName, [className]' [-a filename...] [-b] [-B filename [:arg...]]
[-C] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R
option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename... [class{className:
[mfilename1,mfilename2,...]}...]
mcc -W CTF:archive_name [-a filename...] [-b] [-B filename [:arg...]] [-d
outFolder] [-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R
option] [-S] [-v] [-w option [:msg]] [-Y filename] mfilename... [class{className:
[mfilename1,mfilename2,...]}...]
mcc -W mpsxl:archive_name, [className], [version] -T link:lib [-
replaceBlankWithNaN] [-convertDateToString] [-replaceNaNWithZero] [-
convertNumericToDate] [-a filename...] [-b] [-B filename [:arg...]] [-d outFolder]
[-f filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option]

```

```

[-S] [-v] [-w option [:msg]] [-Y filename] filename... [class{className:
[mfilename1,mfilename2,...]}...]
mcc -H -W mpsxl:archive_name, [className], [version] -T link:lib [-
replaceBlankWithNaN] [-convertDateToString] [-replaceNaNWithZero] [-
convertNumericToDate] [-a filename...] [-b] [-B filename [:arg...]] [-d outFolder] [-f
filename] [-I directory...] [-K] [-M string] [-N] [-p path...] [-R option] [-S] [-v] [-w
option [:msg]] [-Y filename] filename... [class{className: [mfilename...]}...]
mcc -H -W hadoop:archive_name,CONFIG:configFile [-a filename...] [-b] [-B
filename [:arg...]] [-d outFolder] [-f filename] [-I directory...] [-K] [-M string]
[-N] [-p path...] [-R option] [-S] [-v] [-w option [:msg]] [-Y filename] filename...
[class{className: [mfilename1,mfilename2,...]}...]
mcc -?

```

Description

`mcc -m mfilename` compiles the function into a standalone application.

This is equivalent to `-W main -T link:exe`.

`mcc -e mfilename` compiles the function into a standalone application that does not open an MS-DOS[®] command window.

This is equivalent to `-W WinMain -T link:exe`.

`mcc -l mfilename...` compiles the listed functions into a C shared library and generates C wrapper code for integration with other applications.

This is equivalent to `-W lib:libname -T link:lib`.

`mcc -c mfilename...` generates C wrapper code for the listed functions.

This is equivalent to `-W lib:libname -T codegen`.

`mcc -W cpplib:library_name -T link:lib mfilename...` compiles the listed functions into a C++ shared library and generates C++ wrapper code for integration with other applications.

`mcc -W dotnet:assembly_name,className,framework_version,security,remote_type -T link:lib mfilename...` creates a .NET assembly from the specified files.

- *assembly_name* — Specifies the name of the assembly and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the .NET class to be created.
- *framework_version* — Specifies the version of the Microsoft® .NET Framework you want to use to compile the assembly. Specify either:
 - `0.0` — Use the latest supported version on the target machine.
 - *version_major.version_minor* — Use a specific version of the framework.

Features are often version-specific. Consult the documentation for the feature you are implementing to get the Microsoft .NET Framework version requirements.

- *security* — Specifies whether the assembly to be created is a private assembly or a shared assembly.
 - To create a private assembly, specify `Private`.
 - To create a shared assembly, specify the full path to the encryption key file used to sign the assembly.
- *remote_type* — Specifies the remoting type of the assembly. Values are `remote` and `local`.

By default, the compiler generates a single class with a method for each function specified on the command line. You can instruct the compiler to create multiple classes using `class{className:mfilename...}... className` specifies the name of the class to create using *mfilename*.

`mcc -W excel:addin_name,className,version -T link:lib mfilename...`
creates a Microsoft Excel® add-in from the specified files.

- *addin_name* — Specifies the name of the addin and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the *addin_name* as the default.
- *version* — Specifies the version of the add-in specified as *major.minor*.
 - *major* — Specifies the major version number. If you do not specify a version number, `mcc` uses the latest version.
 - *minor* — Specifies the minor version number. If you do not specify a version number, `mcc` uses the latest version.

`mcc -W 'java:packageName,className' mfilename...` creates a Java[®] package from the specified files.

- *packageName* — Specifies the name of the Java package and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the last item in *packageName*.

By default, the compiler generates a single class with a method for each function specified on the command line. You can instruct the compiler to create multiple classes using `class{className:mfilename...}... className` specifies the name of the class to create using *mfilename*.

`mcc -W CTF:archive_name` instructs the compiler to create a deployable archive that is deployable in a MATLAB Production Server[™] instance.

`mcc -W mpsxl:addin_name,className,version input_marshallng_options output_marshallng_options -T link:lib mfilename...` creates an MATLAB Production Server integrated Microsoft Excel add-in from the specified files.

- *addin_name* — Specifies the name of the add-in and its namespace, which is a period-separated list, such as `companyname.groupname.component`.
- *className* — Specifies the name of the class to be created. If you do not specify the class name, `mcc` uses the *addin_name* as the default.
- *version* — Specifies the version of the add-in specified as *major.minor*.
 - *major* — Specifies the major version number. If you do not specify a version number, `mcc` uses the latest version.
 - *minor* — Specifies the minor version number. If you do not specify a version number, `mcc` uses the latest version.
- *input_marshallng_flags* — Specifies options for how data is marshaled between Microsoft Excel and MATLAB.
 - `-replaceBlankWithNaN` — Specifies that a blank in Microsoft Excel is marshaled into NaN in MATLAB. If you do not specify this flag, blanks are marshaled into 0.
 - `-convertDateToString` — Specifies that dates in Microsoft Excel are marshaled into MATLAB strings. If you do not specify this flag, dates are marshaled into MATLAB doubles.

- *output_marshaling_flags* — Specifies options for how data is marshaled between MATLAB and Microsoft Excel.
 - *-replaceNaNWithZero* — Specifies that NaN in MATLAB is marshaled into a 0 in Microsoft Excel. If you do not specify this flag, NaN is marshaled into #QNAN in Visual Basic®.
 - *-convertNumericToDate* — Specifies that MATLAB numeric values are marshaled into Microsoft Excel dates. If you do not specify this flag, Microsoft Excel does not receive dates as output.

`mcc -H -W hadoop:archiveName,CONFIG:configFile` generates deployable archive that can be run as a job by Hadoop.

- *archiveName* — Specifies the name of the generated archive.
- *configFile* — Specifies the path to the Hadoop settings file. See “Hadoop Settings File”.

`mcc -?` displays help.

Tip You can issue the `mcc` command either from the MATLAB command prompt or the DOS or UNIX® command line.

Options

-a Add to Archive

Add files to the deployable archive using

`-a path`

to specify files to be added to the deployable archive. Multiple `-a` options are permitted.

If a file name is specified with `-a`, the compiler looks for these files on the MATLAB path, so specifying the full path name is optional. These files are not passed to `mbuild`, so you can include files such as data files.

If a folder name is specified with the `-a` option, the entire contents of that folder are added recursively to the deployable archive. For example

```
mcc -m hello.m -a ./testdir
```

specifies that all files in `testdir`, as well as all files in subfolders of `testdir`, are added to the deployable archive, and the folder subtree in `testdir` is preserved in the deployable archive.

If a wildcard pattern is included in the file name, only the files in the folder that match the pattern are added to the deployable archive and subfolders of the given path are not processed recursively. For example

```
mcc -m hello.m -a ./testdir/*
```

specifies that all files in `./testdir` are added to the deployable archive and subfolders under `./testdir` are not processed recursively.

```
mcc -m hello.m -a ./testdir/*.m
```

specifies that all files with the extension `.m` under `./testdir` are added to the deployable archive and subfolders of `./testdir` are not processed recursively.

Note: Currently, `*` is the only supported wildcard.

All files added to the deployable archive using `-a` that do not appear on the MATLAB path at the time of compilation causes a path entry to be added to the deployed application's run-time path so that they appear on the path when the deployed code executes.

When files are included, the absolute path for the DLL and header files is changed. The files are placed in the `.\exe_mcr\` folder when the deployable archive is expanded. The file is not placed in the local folder. This folder is created from the deployable archive the first time the application is executed. The `isdeployed` function is provided to help you accommodate this difference in deployed mode.

The `-a` switch also creates a `.auth` file for authorization purposes. It ensures that the executable looks for the DLL- and H-files in the `exe_mcr\exe` folder.

Caution If you use the `-a` flag to include a file that is not on the MATLAB path, the folder containing the file is added to the MATLAB dependency analysis path. As a result, other files from that folder might be included in the compiled application.

Note: If the `-a` flag is used to include custom Java classes, standalone applications work without any need to change the `classpath` as long as the Java class is not a member of a package. The same applies for JAR files. However, if the class being added is a member of a package, the MATLAB code needs to make an appropriate call to `javaaddpath` to update the `classpath` with the parent folder of the package.

-b Generate Excel Compatible Formula Function

Generate a Visual Basic file (`.bas`) containing the Microsoft Excel Formula Function interface to the COM object generated by MATLAB Compiler. When imported into the workbook Visual Basic code, this code allows the MATLAB function to be seen as a cell formula function.

-B Specify Bundle

Replace the file on the `mcc` command line with the contents of the specified file. Use

```
-B filename[:<a1>,<a2>,...,<an>]
```

The bundle `filename` should contain only `mcc` command-line options and corresponding arguments and/or other file names. The file might contain other `-B` options. A bundle can include replacement parameters for compiler options that accept names and version numbers. See “Using Bundles to Build MATLAB Code”.

-C Do Not Embed Deployable Archive

Do not embed the deployable archive in binaries.

-d Specified Folder for Output

Place output in a specified folder. Use

```
-d outFolder
```

to direct the generated files to *outFolder*.

-f Specified Options File

Override the default options file with the specified options file. Use

`-f filename`

to specify `filename` as the options file when calling `mbuild`. This option lets you use different ANSI compilers for different invocations of the compiler. This option is a direct pass-through to `mbuild`.

-g Generate Debugging Information

Include debugging symbol information for the C/C++ code generated by MATLAB Compiler SDK™. It also causes `mbuild` to pass appropriate debugging flags to the system C/C++ compiler. The debug option lets you backtrace up to the point where you can identify if the failure occurred in the initialization of MATLAB Runtime, the function call, or the termination routine. This option does not let you debug your MATLAB files with a C/C++ debugger.

-G Debug Only

Same as `-g`.

-I Add Folder to Include Path

Add a new folder path to the list of included folders. Each `-I` option adds a folder to the beginning of the list of paths to search. For example,

```
-I <directory1> -I <directory2>
```

sets up the search path so that `directory1` is searched first for MATLAB files, followed by `directory2`. This option is important for standalone compilation where the MATLAB path is not available.

If used in conjunction with the `-N` option, the `-I` option will add the folder to the compilation path in the same position where it appeared in the MATLAB path rather than at the head of the path.

-K Preserve Partial Output Files

Direct `mcc` not to delete output files if the compilation ends prematurely, due to error.

The default behavior of `mcc` is to dispose of any partial output if the command fails to execute successfully.

-M Direct Pass Through

Define compile-time options. Use

`-M string`

to pass `string` directly to `mbuild`. This provides a useful mechanism for defining compile-time options, e.g., `-M "-Dmacro=value"`.

Note: Multiple `-M` options do not accumulate; only the rightmost `-M` option is used.

-N Clear Path

Passing `-N` effectively clears the path of all folders except the following core folders (this list is subject to change over time):

- `matlabroot\toolbox\matlab`
- `matlabroot\toolbox\local`
- `matlabroot\toolbox\compiler`

It also retains all subfolders of the above list that appear on the MATLAB path at compile time. Including `-N` on the command line lets you replace folders from the original path, while retaining the relative ordering of the included folders. All subfolders of the included folders that appear on the original path are also included. In addition, the `-N` option retains all folders that you included on the path that are not under `matlabroot\toolbox`.

When using the `-N` option, use the `-I` option to force inclusion of a folder, which will be placed at the head of the compilation path. Use the `-p` option to conditionally include folders and their subfolders; if they are present in the MATLAB path, they will appear in the compilation path, in the same order.

-o Specify Output Name

Specify the name of the final executable (standalone applications only). Use

`-o outputfile`

to name the final executable output of MATLAB Compiler. A suitable, possibly platform-dependent, extension is added to the specified name (e.g., `.exe` for Windows® standalone applications).

-p Add Folder to Path

Use in conjunction with the required option `-N` to add specific folders (and subfolders) under `matlabroot\toolbox` to the compilation MATLAB path in the same order in which they appear in the MATLAB path. Use the syntax

`-N -p directory`

where `directory` is the folder to be included. If `directory` is not an absolute path, it is assumed to be under the current working folder.

- If a folder is included with `-p` that is on the original MATLAB path, the folder and all its subfolders that appear on the original path are added to the compilation path in the same order.
- If a folder is included with `-p` that is not on the original MATLAB path, that folder is ignored. (You can use `-I` to force its inclusion.)

-R Run-Time

Provides MATLAB Runtime options. The syntax is as follows:

`-R option`

Option	Description	Target
<code>-logfile,</code>	Specify a log file name.	MATLAB Compiler MATLAB Compiler SDK
<code>-nodisplay</code>	Suppress the MATLAB <code>nodisplay</code> runtime warning.	MATLAB Compiler MATLAB Compiler SDK
<code>-nojvm</code>	Do not use the Java Virtual Machine (JVM).	MATLAB Compiler MATLAB Compiler SDK
<code>-startmsg</code>	Customizable user message displayed at initialization time.	MATLAB Compiler Standalone Applications

Option	Description	Target
-complete	Customizable user message displayed when initialization is complete.	MATLAB Compiler Standalone Applications

Caution When running on Mac OS X, if `-nodisplay` is used as one of the options included in `mclInitializeApplication`, then the call to `mclInitializeApplication` must occur before calling `mclRunMain`.

-S Create Singleton MATLAB Runtime Context

The standard behavior for the MATLAB Runtime is that every instance of a class gets its own runtime context. This runtime context includes a global MATLAB workspace for variables such as the path and a base workspace for each function in the class. If multiple instances of a class are created, each instance gets an independent context. This ensures that changes made to the global, or base, workspace in one instance of the class does not affect other instances of the same class.

In a singleton MATLAB Runtime, all instances of a class share the runtime context. If multiple instances of a class are created, they use the runtime context created by the first instance. This saves start up time and some resources. However, any changes made to the global workspace or the base workspace by one instance impacts all of the class instances. For example, if `instance1` creates a global variable `A` in a singleton MATLAB Runtime, the `instance2` will be able to use variable `A`.

Singleton MATLAB Runtime is only supported by the following products on these specific targets:

Target supported by Singleton MATLAB Runtime	Create a Singleton MATLAB Runtime by...
Excel add-in	Default behavior for target is singleton runtime. You do not need to perform other steps.
.NET assembly	Default behavior for target is singleton runtime. You do not need to perform other steps.
COM component	<ul style="list-style-type: none"> Using the Library Compiler app, click Settings and add -S to the Additional flags to pass to mcc field. Using <code>mcc</code> pass the <code>-S</code> flag.
Java packages	

-T Specify Target Stage

Specify the output target phase and type.

Use the syntax `-T target` to define the output type. Target values are as follow.

Target	Description
<code>compile:exe</code>	Generate a C/C++ wrapper file and compile C/C++ files to object form suitable for linking into a standalone application.
<code>compile:lib</code>	Generate a C/C++ wrapper file and compile C/C++ files to object form suitable for linking into a shared library/DLL.
<code>link:exe</code>	Same as <code>compile:exe</code> and links object files into a standalone application.
<code>link:lib</code>	Same as <code>compile:lib</code> and links object files into a shared library/DLL.

-u Register COM Component for the Current User

Register COM component for the current user only on the development machine. The argument applies only for generic COM component and Microsoft Excel add-in targets only.

-v Verbose

Display the compilation steps, including:

- MATLAB Compiler version number
- The source file names as they are processed
- The names of the generated output files as they are created
- The invocation of `mbuild`

The `-v` option passes the `-v` option to `mbuild` and displays information about `mbuild`.

-w Warning Messages

Display warning messages. Use the syntax

-w option [:<msg>]

to control the display of warnings. This table lists the syntaxes.

Syntax	Description
-w list	List all of the possible warning that mcc can generate.
-w enable	Enable complete warnings.
-w disable[:<string>]	Disable specific warnings associated with <string>. See “Warning Messages” for a list of the <string> values. Omit the optional <string> to apply the disable action to all warnings.
-w enable[:<string>]	Enable specific warnings associated with <string>. See “Warning Messages” for a list of the <string> values. Omit the optional <string> to apply the enable action to all warnings.
-w error[:<string>]	Treat specific warnings associated with <string> as an error. Omit the optional <string> to apply the error action to all warnings.
-w off[:<string>] [<filename>]	Turn warnings off for specific error messages defined by <string>. You can also narrow scope by specifying warnings be turned off when generated by specific <filename>s.
-w on[:<string>] [<filename>]	Turn warnings on for specific error messages defined by <string>. You can also narrow scope by specifying warnings be turned on when generated by specific <filename>s.

It is also possible to turn warnings on or off in your MATLAB code.

For example, to turn warnings off for deployed applications (specified using `isdeployed`) in your `startup.m`, you write:

```
if isdeployed
    warning off
end
```

To turn warnings on for deployed applications, you write:

```
if isdeployed
```

```
    warning on  
end
```

-Y License File

Use

```
-Y license.lic
```

to override the default license file with the specified argument.

See Also

`mbuild`

matlab.mapreduce.DeployHadoopMapReducer class

Package: matlab.mapreduce

Configure a MapReduce application for deployment against Hadoop

Description

MapReducer object that represents executing MapReduce on a Hadoop cluster with a deployed MATLAB Runtime.

Construction

`config = matlab.mapreduce.DeployHadoopMapReducer` creates a `matlab.mapreduce.DeployHadoopMapReducer` object that specifies the default properties for Hadoop execution.

Use the resulting object as input to the `mapreducer` function, to specify the configuration properties for Hadoop execution. For deploying a standalone application, pass the `matlab.mapreduce.DeployHadoopMapReducer` object as input to `mapreduce`.

`config = matlab.mapreduce.DeployHadoopMapReducer(Name, Value)` creates a `matlab.mapreduce.DeployHadoopMapReducer` object with properties specified by one or more name-value pair arguments.

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name, Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `'MCRRoot', '/hd-shared/hadoop-2.2.0/MCR/v84'`

'HadoopInstallFolder' — Path to Hadoop installation
character string

Path to Hadoop installation, specified as the comma-separated pair consisting of the `HadoopInstallFolder` and a character string.

The default value of Hadoop install folder is specified by the environment variables in the order of precedence of `MATLAB_HADOOP_INSTALL`, `HADOOP_PREFIX`, and `HADOOP_HOME`.

'HadoopConfigurationFile' — Path to Hadoop application configuration files

character string

Path to Hadoop application configuration files, specified as the comma-separated pair consisting of the `HadoopConfigurationFile` and a character string.

'MCRRoot' — MATLAB Runtime install folder for Hadoop cluster

character string

MATLAB Runtime install folder for Hadoop cluster, specified as the comma-separated pair consisting of the `MCRRoot` and a character string.

`MCRRoot` specifies the MATLAB Runtime install folder used by Hadoop when executing `mapreduce` tasks in Hadoop.

'HadoopProperties' — Map container of name-value pairs

character string | cell array of strings

Map container of name-value pairs, specified as the comma-separated pair consisting of the `HadoopProperties` and a character string or a cell array of strings.

Map container values are passed as inputs to the Hadoop command.

Properties

HadoopInstallFolder — Path to Hadoop installation

character string

Path to Hadoop installation, specified as a character string.

HadoopConfigurationFile — Path to Hadoop application configuration files

character string

Path to Hadoop application configuration files, specified as a character string.

MCRRoot — MATLAB Runtime install folder for Hadoop cluster

character string

MATLAB Runtime install folder for Hadoop cluster, specified as a character string.

MCRRoot specifies the MATLAB Runtime install folder used by Hadoop when executing mapreduce tasks in Hadoop.

HadoopProperties — Map container of name-value pairs

character string | cell array of strings

Map container of name-value pairs, specified as a character string or a cell array of strings.

Map container values are passed as inputs to the Hadoop command.

Examples

Create a Deploy Hadoop MapReducer object

Create and use a `matlab.mapreduce.DeployHadoopMapReducer` object to deploy into a standalone application and deploy against Hadoop.

```
config = matlab.mapreduce.DeployHadoopMapReducer('MCRRoot', ...  
         '/hd-shared/hadoop-2.2.0/MCR/v84');  
mr = mapreducer(config);
```

- “Create Standalone Application to Run Against Hadoop from Command Line” on page 2-2

See Also

mapreduce | mapreducer

hadoopCompiler

Build and package MapReduce applications for deployment against Hadoop

Syntax

```
hadoopCompiler  
hadoopCompiler project_name
```

Description

hadoopCompiler opens the Hadoop compiler app

hadoopCompiler project_name opens the MATLAB compiler with the project preloaded.

Examples

Create a New Hadoop Compiler Project

Open the Hadoop compiler app to create a new project.

```
hadoopCompiler
```

Input Arguments

project_name — name of the project to be compiled
string

Specify the name of a previously saved MATLAB Compiler project. The project must be on the current path.

See Also

deploytool | mcc

mapreducer

Define deployed execution for mapreduce

Use this function with MATLAB Compiler to specify information about the execution environment for standalone applications that execute against Hadoop.

Syntax

```
mapreducer(config)
mr = mapreducer(config)
```

Description

`mapreducer(config)` specifies execution environment. When deploying a standalone application against Hadoop, `config` is an object of `matlab.mapreduce.DeployHadoopMapReducer` class.

`mr = mapreducer(config)` returns a `MapReducer` object to specify the execution environment. You can define `MapReducer` objects, allowing you to swap execution environments by passing one as an input argument to `mapreduce`.

Examples

- “Create Standalone Application to Run Against Hadoop from Command Line” on page 2-2

Input Arguments

config — **mapreducer object for running in deployed environment**

`matlab.mapreduce.DeployHadoopMapReducer` object

`mapreducer` object for running in deployed environment, specified as a `matlab.mapreduce.DeployHadoopMapReducer` object.

```
Example: config =  
mapreducer(matlab.mapreduce.DeployHadoopMapReducer('MCRRoot', '/hd-  
shared/hadoop-2.2.0/MCR/v84'))
```

Output Arguments

mr — Execution environment for MapReduce

MapReducer object

Execution environment for `mapreduce`, returned as a `MapReducer` object.

More About

Tips

- `mapreducer` and `mapreducer(0)` enables different configurations based on the products you have. In MATLAB, the `mapreduce` function automatically runs using a `SerialMapReducer`. For more information, see `mapreducer`.

If you have Parallel Computing Toolbox™, see the function reference page for `mapreducer` for additional information.

See Also

Functions

`gcmr` | `mapreduce`

Classes

`matlab.mapreduce.DeployHadoopMapReducer`

Apps

Hadoop Compiler

Package MATLAB programs for deployment to Hadoop clusters as MapReduce programs

Description

The **Hadoop Compiler** packages MATLAB functions into applications for deployment to Hadoop clusters as MapReduce programs.

Open the Hadoop Compiler App

- MATLAB Toolstrip: On the **Apps** tab, under **Application Deployment**, click the app icon.
- MATLAB command prompt: Enter `hadoopCompiler`.

Examples

- “Package Deployable Archive to Run Against Hadoop with Hadoop Compiler App”

More About

- “Large Files and Big Data”

Parameters

Map Function

Specify the MATLAB function for the mapper.

Reduce Function

Specify the MATLAB function for the reducer.

Input Types

Specify how the MATLAB Runtime is packaged with the application.

Default: tabulartext

Settings

binary

The input is the result of a previous MapReduce job that was saved as a sequence file.

tabulartext

The input is a formatted text file.

Output Types

Specify how the MATLAB Runtime is packaged with the application.

Default: binary

Settings

binary

The output is saved as a sequence file.

tabulartext

The output is stored as a formatted text file.

Settings

Specify the output folders for the packaged code.

Default:

- Testing folder: `for_testing`
- Redistribution folder: `for_redistribution`

Additional Configuration File Content

Specifies additional parameters to configure how Hadoop runs the job. See “Hadoop Settings File”.

Data Store File

Specify the data store for the job to use.

Files Required for MapReduce Job to Run

Specify the MATLAB files and data files that the MapReduce job requires to run. The listed files are packaged into the generated archive.

Default: The list of files generated by the built-in dependency analysis tool.

Programmatic Use

```
hadoopCompiler
```