MATLAB® Compiler™

Hadoop® Integration Guide

MATLAB®

R2017a

MathWorks®

MATLAB® Compiler™ Hadoop® Integration Guide

Revision History

October 2014	Online only	New for Version 5.2 (Release 2014b)
March 2015	Online only	Revised for Version 6.0 (Release 2015a)
September 2015	Online only	Revised for Version 6.1 (Release 2015b)
October 2015	Online only	Rereleased for Version 6.0.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 6.2 (Release 2016a)
September 2016	Online Only	Revised for Version 6.3 (Release 2016b)
March 2017	Online only	Revised for Version 6.4 (Release R2017a)

Contents

Deployable Archives

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® 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. Move your example code to a new working folder for deployment. The new working folder on the path ensures that the files are accessible by MATLAB Compiler™.

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

---

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

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

   b. 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`.

      For information on installing the MATLAB Runtime, “Install and Configure the MATLAB Runtime”.

   c. Copy the `airlinesmall.csv` into Hadoop Distributed File System (HDFS™) folder `/datasets/airlinemod`.

   d. 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” (MATLAB).

e. 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” (MATLAB).

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

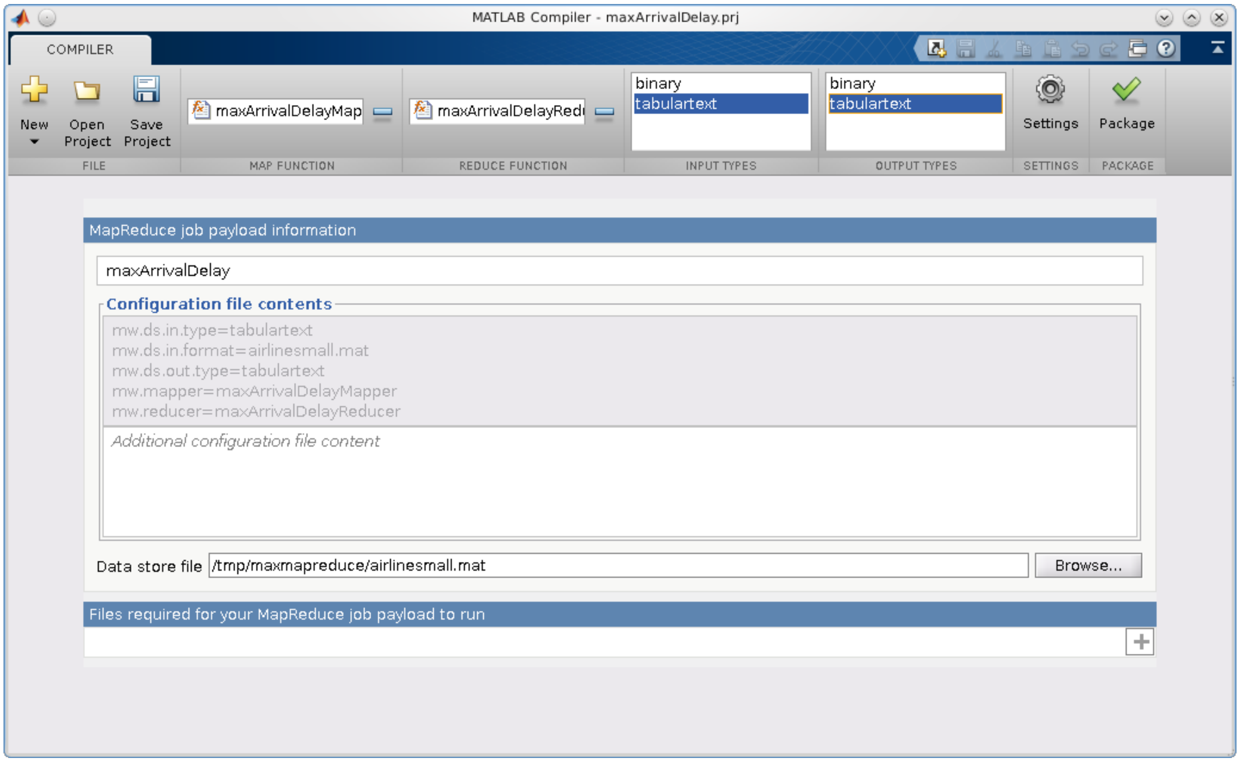
```
ds = datastore('airlinesmall.csv','TreatAsMissing','NA',...
     'SelectedVariableNames','ArrDelay','ReadSize',1000);

save('airlinesmall.mat','ds')
```

For more information, “Getting Started with Datastore” (MATLAB)

3. 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
```



maxArrivalDelay

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

Additional configuration file content

Data store file /tmp/maxmapreduce/airlinesmall.mat

4 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`.

5 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`.

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

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

8 Rename the application name to `maxArrivalDelay`.

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

10 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`.

11 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
```

12 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” (MATLAB).

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 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. Move your example code to a new working folder for deployment. The new working folder on the path ensures that the files are accessible by MATLAB Compiler.

Note: Deployable archive that runs against Hadoop using Hadoop Compiler app is supported only on Linux.

---

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

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

   b 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`.

      Download the MATLAB Runtime from the website at http://www.mathworks.com/products/compiler/mcr.

   c Copy the `airlinesmall.csv` into Hadoop Distributed File System (HDFS) folder `/datasets/airlinemod`.

   d 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” (MATLAB).

   e 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” (MATLAB).

2 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” (MATLAB)

3 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.

4 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`.

5 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
```

6 Visualize and plot the results.

```
ds = datastore('hdfs://hadoop01/user/username/myresults/part*',...
     'Type', '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” (MATLAB).

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 you how to create a standalone MapReduce application from the command line and deploy it against a Hadoop cluster.

This example uses the airline dataset, `airlinesmall.csv`, which contains departure and arrival information about individual airline flights. The goal of this example is to compute the maximum arrival delay of an airline in the dataset. The dataset is available in the *`matlabroot`*`/toolbox/matlab/demos` folder.

Supported Platform: Linux only.

Prerequisites

- Start this example by creating a new work folder that is visible to the MATLAB search path.
- Before starting MATLAB, at a Terminal window, set the environment variable `HADOOP_PREFIX` to point to the Hadoop installation folder. For example:

Shell	Command
CSH	`% setenv HADOOP_PREFIX /usr/lib/hadoop`
BASH	`$ export HADOOP_PREFIX=/usr/lib/hadoop`

Note: This example uses `/usr/lib/hadoop` as directory where Hadoop is installed. Your Hadoop installation directory maybe different.

If you forget setting the `HADOOP_PREFIX` environment variable prior to starting MATLAB, set it up using the MATLAB function `setenv` at the MATLAB command prompt as soon as you start MATLAB. For example:

```
setenv('HADOOP_PREFIX','/usr/lib/hadoop')
```

- Install the MATLAB Runtime in a folder that is accessible by every worker node in the Hadoop cluster. This example uses `/usr/local/MATLAB/MATLAB_Runtime/v91` as the location of the MATLAB Runtime folder.

  If you don't have the MATLAB Runtime, you can download it from the website at: http://www.mathworks.com/products/compiler/mcr.

- Create the directory `/user/<username>/datasets` on HDFS and copy the file `airlinesmall.csv` to that directory. Here *`<username>`* refers to your user name in HDFS.

```
$ ./hadoop fs -copyFromLocal airlinesmall.csv hdfs://hadoopfs:54310/user/<username>
```

- Copy the map function `maxArrivalDelayMapper.m` from *`matlabroot`*`/toolbox/matlab/demos` folder to the work folder.

  `maxArrivalDelayMapper.m`

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

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

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

  `maxArrivalDelayReducer.m`

  ```
  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” (MATLAB).

Procedure

1. Start MATLAB and verify that the `HADOOP_PREFIX` environment variable has been set. At the command prompt, type:

   ```
   >> getenv('HADOOP_PREFIX')
   ```

   If `ans` is empty, review the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.
2. Create a new MATLAB script with the name `depMapRedStandAlone.m`. You will add the code listed in the steps listed below to this script file.
3. Create a `datastore` that points to the airline data in Hadoop Distributed File System (HDFS) .

```
ds = datastore('hdfs://hadoopglnxa64:54310/user/username/datasets/airlinesmall.csv'
'TreatAsMissing','NA',...
'SelectedVariableNames',{'UniqueCarrier','ArrDelay'});
```

For more information, see “Read Remote Data” (MATLAB).

4 Configure the application for deployment against Hadoop with default settings.

```
config = matlab.mapreduce.DeployHadoopMapReducer;
```

The class `matlab.mapreduce.DeployHadoopMapReducer` can be used to configure a standalone application based on the Hadoop environment where it is going to be deployed.

For example, if you want to specify the location of the MATLAB Runtime on each of the worker nodes on the cluster, include a line of code similar to this:

```
config = matlab.mapreduce.DeployHadoopMapReducer('MCRRoot','/opt/MATLAB/MATLAB_Runt
```

In this scenario, we assume that the MATLAB Runtime is installed in a non-default location such as `/opt/MATLAB/MATLAB_Runtime` on the worker nodes.

For information on specifying additional cluster specific properties, see `matlab.mapreduce.DeployHadoopMapReducer`.

Note: Specifying a MATLAB Runtime location as part of the class `matlab.mapreduce.DeployHadoopMapReducer` will override any MATLAB Runtime location specified during the execution of the standalone application.

5 Define the execution environment using the `mapreducer`.

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

6 Apply the `mapreduce` function.

```
result = mapreduce(...
    ds,...
    @maxArrivalDelayMapper,@maxArrivalDelayReducer,...
    mr,...
    'OutputType','Binary', ...
    'OutputFolder','hdfs://hadoopglnxa64:54310/user/username/results/myresults');
```

Note: An HDFS directory such as `.../myresults` can be written to only once. If you plan on running your standalone application multiple times against the Hadoop cluster, make sure you delete the `.../myresults` directory on HDFS prior to each execution. Another option is to change the name of the `.../myresults` directory in the MATLAB code and recompile the application.

7. Read the result from the resulting datastore.

   ```
   myAppResult = readall(result)
   ```
8. Use the `mcc` command with the `-m` flag to create a standalone application.

   ```
   mcc -m depMapRedStandAlone.m
   ```

   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.
9. Run the standalone application from a Linux shell using the following command:

   ```
   $ ./run_depMapRedStandAlone.sh /usr/local/MATLAB/MATLAB_Runtime/v91
   ```

   `/usr/local/MATLAB/MATLAB_Runtime/v91` is an argument indicating the location of the MATLAB Runtime.

   Prior to executing the above command, verify that the `HADOOP_PREFIX` environment variable is set in the Terminal by typing:

   ```
   $ echo $HADOOP_PREFIX
   ```
   If `echo` comes up empty, see the **Prerequisites** section above to see how you can set the `HADOOP_PREFIX` environment variable.

   Your application will fail to execute if the `HADOOP_PREFIX` environment variable is not set.
10. You will see the following output:

   ```
   myAppResult = 

              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 standalone applications that run against Hadoop. For more information, see “Build Effective Algorithms with MapReduce” (MATLAB).

Complete code for the standalone application `depMapRedStandAlone` can be found here:

depMapRedStandAlone.m

```
%% Create datastore
ds = datastore(...
 'hdfs://hadoopglnxa64:54310/user/username/datasets/airlinesmall.csv',...
 'TreatAsMissing','NA',...
 'SelectedVariableNames',{'UniqueCarrier','ArrDelay'});

%% Configure application for deployment against Hadoop with default settings
config = matlab.mapreduce.DeployHadoopMapReducer;

%% Define the execution environment
mr = mapreducer(config);

%% Apply the mapreduce function
result = mapreduce(...
    ds,...
    @maxArrivalDelayMapper,@maxArrivalDelayReducer,...
    mr,...
    'OutputType','Binary', ...
    'OutputFolder','hdfs://hadoopglnxa64:54310/user/username/results/myresults');

%% Read the result from the resulting datastore
myAppResult = readall(result)
```

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”

Hadoop Configuration

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`

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

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

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
character array or string

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

Introduced in R2006b

mcc

Compile MATLAB functions for deployment

Syntax

```
mcc options mfilename1,...,mfilenameN

mcc -m options mfilename
mcc -e options mfilename

mcc -W 'excel:addin_name,className,version' -T link:lib options
mfilename1,...,mfilenameN

mcc -H -W hadoop:archiveName,CONFIG:configFile
```

Description

`mcc options mfilename1,...,mfilenameN` compiles the functions as specified by the options.

The options used depend on the intended results of the compilation. For information on compiling:

- C/C++ shared libraries, .NET assemblies, Java® packages, or Python® packages see `mcc` for MATLAB Compiler SDK™
- MATLAB Production Server™ deployable archives or Excel® add-ins for MATLAB Production Server see `mcc` for MATLAB Compiler SDK

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

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

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

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

`mcc -W 'excel:`*`addin_name`*`,`*`className`*`,`*`version`*`' -T link:lib options mfilename1,...,mfilenameN` 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. If specified, *`className`*, needs to be different from *`mfilename`*.
- *`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.

Note: Excel add-ins can only be created in MATLAB running on Windows®.

Note: Remove the single quotes around `'excel:addin_name,className,version'` when executing the `mcc` command from a DOS prompt.

`mcc -H -W hadoop:`*`archiveName`*`,CONFIG:`*`configFile`* generates a 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” on page 3-3.

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

Examples

Compile a standalone application

```
mcc -m magic.m
```

Compile a standalone Windows application

Compile a standalone application that does not open a command prompt on Windows.

```
mcc -e magic.m
```

Compile an Excel add-in

```
mcc -W 'excel:myAddin,myClass,1.0' -T link:lib magic.m
```

Input Arguments

`mfilename` — File to be compiled
filename

File to be compiled specified as a character vector.

`mfilename1,...,mfilenameN` — Files to be compiled
list of filenames

One, or more, files to be compiled, specified as a comma-separated list of filenames.

`options` — Options for customizing the output
-a | -b | -B | -C | -d | -f | -g | -G | -I | -K | -m | -M | -N | -o | -p | -R | -S | -T | -u | -v | -w | -W | -Y

Options for customizing the output, specified as a list of character vectors.

- `-a`

  Add files to the deployable archive using `-a path` to specify the files to be added. 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 its subfolders, are added to the deployable archive. The folder subtree in `testdir` is preserved in the deployable archive.

If the filename includes a wildcard pattern, 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: `*` is the only supported wildcard.

When you add files to the archive using `-a` that do not appear on the MATLAB path at the time of compilation, a path entry is added to the application's run-time path so that they appear on the path when the deployed code executes.

When you include files, the absolute path for the DLL and header files changes. The files are placed in the `.\`*`exe_mcr`*`\` folder when the 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 you use the `-a` flag 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 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`

  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” (MATLAB Compiler SDK).
- `-C`

  Do not embed the deployable archive in binaries.
- `-d`

  Place output in a specified folder. Use

  ```
  -d outFolder
  ```

to direct the generated files to *`outFolder`*.

- `-f`

  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`, `-G`

  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.
- `-I`

  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 adds 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`

  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 `mcc` to generate a standalone application.

- `-M`

  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`

  Passing `-N` 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`

  Passing `-N` also retains all subfolders in this 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 is 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 appear in the compilation path in the same order.

- `-o`

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

  ```
  -o outputfile
  ```

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

- `-p`

  Use in conjunction with the option `-N` to add specific folders and subfolders under *`matlabroot`*`\toolbox` to the compilation MATLAB path. The files are added 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`

  Provides MATLAB Runtime options. This option is only relevant when building standalone applications using MATLAB Compiler. The syntax is as follows:

  ```
  -R option
  ```

Option	Description	Target
`-logfile,`	Specify a log file name.	MATLAB Compiler
`-nodispla`	Suppress the MATLAB `nodisplay` run-time warning.	MATLAB Compiler
`-nojvm`	Do not use the Java Virtual Machine (JVM).	MATLAB Compiler
`-startmsg`	Customizable user message displayed at initialization time.	MATLAB Compiler Standalone Applications
`-complete`	Customizable user message displayed when initialization is complete.	MATLAB Compiler

Option	Description	Target
		Standalone Applications

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

Note: If you specify the `-R` option for libraries created from MATLAB Compiler SDK, `mcc` will still compile without errors and generate the results. But the `-R` option doesn't apply to these libraries and won't do anything.

- `-S`

  The standard behavior for the MATLAB Runtime is that every instance of a class gets its own MATLAB Runtime context. The 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 context. If multiple instances of a class are created, they use the context created by the first instance. This saves startup time and some resources. However, any changes made to the global workspace or the base workspace by one instance impacts all class instances. For example, if `instance1` creates a global variable `A` in a singleton MATLAB Runtime, then `instance2` can 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 MATLAB Runtime. You do not need to perform other steps.
.NET assembly	Default behavior for target is singleton MATLAB Runtime. You do not need to perform other steps.

Target supported by Singleton MATLAB Runtime	Create a Singleton MATLAB Runtime by....
COM component	• Using the Library Compiler app, click **Settings** and add `-S` to the **Additional parameters passed to MCC** field. • Using `mcc`, pass the `-S` flag.
Java package	

- `-T`

  Specify the output target phase and type.

  Use the syntax `-T` *`target`* to define the output type.

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

- `-u`

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

- `-v`

  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`

  Display warning messages. Use the syntax

  ```
  -w option [:<msg>]
  ```

  to control the display of warnings.

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.

You can also 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
```

- `-W`

  Control the generation of function wrappers. Use the syntax

  ```
  -W type
  ```

  to control the generation of function wrappers for a collection of MATLAB files generated by the compiler. You provide a list of functions and the compiler generates the wrapper functions and any appropriate global variable definitions.
- `-Y` Use

  ```
   -Y license.lic
  ```

  to override the default license file with the specified argument.

Note: The `-Y` flag works only with the command-line mode.

```
>>!mcc -m foo.m -Y license.lic
```

See Also

Introduced before R2006a

matlab.mapreduce.DeployHadoopMapReducer class

Package: matlab.mapreduce

Configure a MapReduce application for deployment against Hadoop

Description

A `DeployHadoopMapReducer` object represents executing MapReduce on a Hadoop cluster with 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`.

`'HadoopInstallFolder'` — Path to Hadoop installation
character vector

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

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 vector

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

`'MCRRoot'` — MATLAB Runtime install folder for Hadoop cluster

character vector

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

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

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

`'HadoopProperties'` — Job or application specific Hadoop configuration properties

containers.Map

A `containers.Map` object of name-value pairs that specify Hadoop configuration properties for a specific job or application. Name-value pairs must be specified as character vectors.

The properties specified in the `containers.Map` object are passed as a [GENERIC_OPTION] consisting of name-value pairs signaled by a `-D` flag to the `hadoop` shell command.

Example:

```
setenv('HADOOP_PREFIX', '/share/hadoop/a1.2.1') % replace with your Hadoop install loca
name = {'mapreduce.map.maxattempts','mapreduce.job.reduces'};
value = {'2','1'};
prop = containers.Map(name,value);
obj = matlab.mapreduce.DeployHadoopMapReducer('HadoopProperties', prop)
```

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

See Also

`mapreduce` | `mapreducer`

Topics

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

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

character array or string

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

See Also

See Also

`deploytool` | `mcc`

Introduced in R2014b

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 a mapreducer object in deployed mode

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

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.

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

See Also

Functions
`gcmr` | `mapreduce`

Classes
matlab.mapreduce.DeployHadoopMapReducer

Topics
"Create Standalone Application to Run Against Hadoop from Command Line" on page 2-2

Introduced in R2014b

5

Apps

Hadoop Compiler

Package MATLAB programs for deployment to Hadoop clusters as MapReduce programs

Description

The **Hadoop Compiler** app 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" on page 1-2

Parameters

`map function` — function for mapper
character vector

Function for the mapper as a character vector.

`reduce function` — function for reducer
character vector

Function for the reducer as a character vector.

`datastore file` — input file for MapReduce
character vector

Input file for MapReduce as a character vector.

`output types` — format of output
keyvalue (default) | tabulartext

Format of MapReduce output as a character vector.

`additional configuration file content` — additional parameters configuring how Hadoop runs job
character vector

Additional parameters to configure how Hadoop runs the job as a character vector. See "Hadoop Settings File" on page 3-3.

`files required for your MapReduce job payload to run` — files that must be included with generated artifacts
list of files

Files that must be included with generated artifacts as a list of files.

Settings

`Additional parameters passed to MCC` — flags controlling the behavior of the compiler
character vector

Flags controlling the behavior of the compiler as a character vector.

`testing files` — folder where files for testing are stored
character vector

Folder where files for testing are stored as a character vector.

`packaged files` — folder where generated artifacts are stores
character vector

Folder where generated artifacts are stored as a character vector.

Programmatic Use

`hadoopCompiler`

See Also

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

Introduced in R2014b