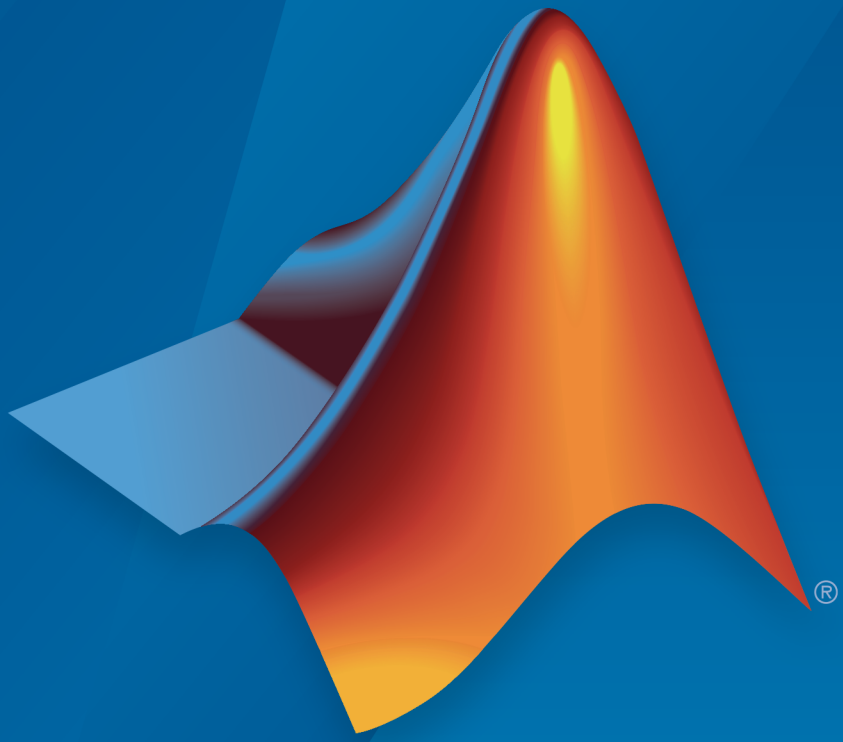
# Parallel Computing Toolbox™ Release Notes



MATLAB®

MathWorks®

# 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

*Parallel Computing Toolbox™ Release Notes*

# Contents

## R2015a

## R2014b



## R2014a

## R2013b

## R2013a

## R2011b

## R2011a



## R2010b

## R2010a

## R2009b



## R2009a

## R2008b



## R2008a

## R2007b



## R2007a

## R2006b

# R2015a

## Support for mapreduce function on any cluster that supports parallel pools

You can now run parallel `mapreduce` on any cluster that supports a parallel pool. For more information, see "Run mapreduce on a Parallel Pool".

## Sparse arrays with GPU-enabled functions

This release supports sparse arrays on a GPU. You can create a sparse gpuArray either by calling `sparse` with a gpuArray input, or by calling `gpuArray` with a sparse input. The following functions support sparse gpuArrays.

```
classUnderlying          isfloat                  nnz
conj                     isinteger                numel
ctranspose               islogical                nzmax
end                      isnumeric                real
find                     isreal                   size
full                     issparse                 sparse
gpuArray.speye           length                   spones
imag                     mtimes                   transpose
isaUnderlying            ndims
isempty                  nonzeros
```

Note the following for some of these functions:

- `gpuArray.speye` is a static constructor method.
- `sparse` supports only single-argument syntax.
- `mtimes` does not support the case of full-matrix times a sparse-matrix.

For more information on this topic, see "Sparse Arrays on a GPU".

A new C function, `mxGPUIsSparse`, is available for the MEX interface, to query whether a gpuArray is sparse or not. However, even though the MEX interface can query properties of a sparse gpuArray, its functions cannot access sparse gpuArray elements.

## Additional GPU-enabled MATLAB functions

The following functions are new in their support for gpuArrays:

| | | |
|---|---|---|
| `cdf2rdf` | `istril` | `polyarea` |
| `gpuArray.freqspace` | `istriu` | `polyder` |
| `histcounts` | `legendre` | `polyfit` |
| `idivide` | `nonzeros` | `polyint` |
| `inpolygon` | `nthroot` | `polyval` |
| `isdiag` | `pinv` | `polyvalm` |
| `ishermitian` | `planerot` | |
| `issymmetric` | `poly` | |

Note the following for some of these functions:

- `gpuArray.freqspace` is a static constructor method.

For a list of MATLAB® functions that support gpuArray, see "Run Built-In Functions on a GPU ".

## `pagefun` support for `mrdivide` and `inv` functions on GPUs

For gpuArray inputs, `pagefun` is enhanced to support:

- `@inv`
- `@mrdivide` (for square matrix divisors of sizes up to 32-by-32)

## Enhancements to GPU-enabled linear algebra functions

Many of the linear algebra functions that support gpuArrays are enhanced for improved performance. Among those functions that can exhibit improved performance are `svd`, `null`, `eig` (for nonsymmetric input), and `mtimes` (for inner products).

## Parallel data reads from a datastore with MATLAB `partition` function

The `partition` function can perform a parallel read and partition of a "Datastore". For more information, see `partition` and `numpartitions`. See also "Partition a Datastore in Parallel".

## Using DNS for cluster discovery

In addition to multicast, the discover cluster functionality of Parallel Computing Toolbox™ can now use DNS to locate MATLAB job scheduler (MJS) clusters. For

information about cluster discovery, see “Discover Clusters”. For information about configuring and verifying the required DNS SRV record on your network, see “DNS SRV Record”.

## MS-MPI support for local and MJS clusters

On 64-bit Windows® platforms, Microsoft® MPI (MS-MPI) is now the default MPI implementation for local clusters on the client machine.

For MATLAB job scheduler (MJS) clusters on Windows platforms, you can use MS-MPI by specifying the `-useMSMPI` flag with the `startjobmanager` command.

## Ports and sockets in `mdce_def` file

The following parameters are new to the `mdce_def` file for controlling the behavior of MATLAB job scheduler (MJS) clusters.

- `ALL_SERVER_SOCKETS_IN_CLUSTER` — This parameter controls whether all client connections are outbound, or if inbound connections are also allowed.
- `JOBMANAGER_PEERSESSION_MIN_PORT`, `JOBMANAGER_PEERSESSION_MAX_PORT` — These parameters set the range of ports to use when `ALL_SERVER_SOCKETS_IN_CLUSTER` = `true`.
- `WORKER_PARALLELPOOL_MIN_PORT`, `WORKER_PARALLELPOOL_MAX_PORT` — These parameters set the range of ports to use on worker machines for parallel pools.

For more information and default settings for these parameters, see the appropriate `mdce_def` file for your platform:

- *`matlabroot`*`\toolbox\distcomp\bin\mdce_def.bat` (Windows)
- *`matlabroot`*`/toolbox/distcomp/bin/mdce_def.sh` (UNIX®)

## Compatibility Considerations

By default in this release, `ALL_SERVER_SOCKETS_IN_CLUSTER` is `true`, which makes all connections outbound from the client. For pre-R2015a behavior, set its value to `false`, which also initiates a set of inbound connections to the client from the MJS and workers.

## Improved profiler accuracy for GPU code

The MATLAB profiler now reports more accurate timings for code running on a GPU. For related information, see “Measure Performance on the GPU”.

## Upgraded CUDA Toolkit version

The parallel computing products are now using CUDA® Toolkit version 6.5. To compile CUDA code for CUDAKernel or CUDA MEX files, you must use toolkit version 6.5.

## Discontinued support for GPU devices on 32-bit Windows computers

This release no longer supports GPU devices on 32-bit Windows machines.

## Compatibility Considerations

GPU devices on 32-bit Windows machines are not supported in this release. Instead, use GPU devices on 64-bit machines.

## Discontinued support for parallel computing products on 32-bit Windows computers

In a future release, support will be removed for Parallel Computing Toolbox and MATLAB Distributed Computing Server™ on 32-bit Windows machines.

## Compatibility Considerations

Parallel Computing Toolbox and MATLAB Distributed Computing Server are still supported on 32-bit Windows machines in this release, but parallel language commands can generate a warning. In a future release, support will be completely removed for these computers, at which time it will not be possible to install the parallel computing products on them.

## `matlabpool` function removed

The `matlabpool` function has been removed.

## Compatibility Considerations

Calling `matlabpool` now generates an error. You should instead use parpool to create a parallel pool.

## Upgrade parallel computing products together

This version of Parallel Computing Toolbox software is accompanied by a corresponding new version of MATLAB Distributed Computing Server software.

## Compatibility Considerations

As with every new release, if you are using both parallel computing products, you must upgrade Parallel Computing Toolbox and MATLAB Distributed Computing Server together. These products must be the same version to interact properly with each other.

Jobs created in one version of Parallel Computing Toolbox software will not run in a different version of MATLAB Distributed Computing Server software, and might not be readable in different versions of the toolbox software. The job data stored in the folder identified by `JobStorageLocation` (formerly `DataLocation`) might not be compatible between different versions of MATLAB Distributed Computing Server. Therefore, `JobStorageLocation` should not be shared by parallel computing products running different versions, and each version on your cluster should have its own `JobStorageLocation`.

# R2014b

## Parallelization of `mapreduce` on local workers

If you have Parallel Computing Toolbox installed, and your default cluster profile specifies a local cluster, then execution of mapreduce opens a parallel pool and distributes tasks to the pool workers.

---

**Note** If your default cluster profile specifies some other cluster, the `mapreduce` function does not use a parallel pool.

---

For more information, see Run mapreduce on a Local Cluster.

## Additional GPU-enabled MATLAB functions, including `accumarray`, `histc`, `cummax`, and `cummin`

The following functions are new in their support for gpuArrays:

```
accumarray          cummax              psi
acosd               cummin              rgb2hsv
acotd               del2                roots
acscd               factorial           secd
asecd               gammainc            sind
asind               gammaincinv         sph2cart
atan2d              gradient            subspace
atand               hankel              superiorfloat
betainc             histc               swapbytes
betaincinv          hsv2rgb             tand
cart2pol            isaUnderlying       toeplitz
cart2sph            median              trapz
compan              mode                typecast
corrcoef            nextpow2            unwrap
cosd                null                vander
cotd                orth
cscd                pol2cart
```

Note the following for some of these functions:

- The first input argument to `gammainc` cannot contain any negative elements.

For a list of MATLAB functions that support gpuArray, see Run Built-In Functions on a GPU.

## `pagefun` support for `mldivide` on GPUs

For gpuArray inputs, pagefun is enhanced to support `@mldivide` for square matrix divisors of sizes up to 32-by-32.

## Additional MATLAB functions for distributed arrays, including `fft2`, `fftn`, `ifft2`, `ifftn`, `cummax`, `cummin`, and `diff`

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

```
besselh
besseli
besselj
besselk
bessely
beta
betainc
betaincinv
betaln
cart2pol
cart2sph
compan
corrcoef
cov
cummax
cummin
diff
erf
erfc
erfcinv
erfcx
erfinv
fft2
fftn
gamma
gammainc
gammaincinv
gammaln
hankel
hsv2rgb
ifft
ifft2
ifftn
isfloat
isinteger
islogical
isnumeric
median
mode
pol2cart
psi
rgb2hsv
sph2cart
std
toeplitz
trapz
unwrap
vander
var
```

Note the following for some of these functions:

- `isfloat`, `isinteger`, `islogical`, and `isnumeric` now return results based on classUnderlying of the distributed array.

For a list of MATLAB functions that support distributed arrays, see MATLAB Functions on Distributed and Codistributed Arrays.

## Data Analysis on Hadoop clusters using `mapreduce`

Parallel Computing Toolbox and MATLAB Distributed Computing Server support the use of Hadoop® clusters for the execution environment of `mapreduce` applications. For more information, see:

- Configure a Hadoop Cluster
- Run mapreduce on a Hadoop Cluster

## Discover Clusters Supports Microsoft Windows HPC Server for Multiple Releases

The Discover Clusters dialog can now list Microsoft Windows HPC Server clusters for different releases of parallel computing products. For more information about cluster discovery, see Discover Clusters.

For this functionality, the HPC Server client utilities must be installed on the client machine from which you are discovering. See Configure Client Computer for HPC Server.

## Upgraded CUDA Toolkit Version

The parallel computing products are now using CUDA Toolkit version 6.0. To compile CUDA code for CUDAKernel or CUDA MEX files, you must use toolkit version 6.0 or earlier.

## Discontinued Support for GPU Devices of Compute Capability 1.3

This release no longer supports GPU devices of compute capability 1.3.

## Compatibility Considerations

This release supports only GPU devices of compute capability 2.0 or greater.

## Discontinued Support for GPU Devices on 32-Bit Windows Computers

In a future release, support for GPU devices on 32-bit Windows machines will be removed.

## Compatibility Considerations

GPU devices on 32-bit Windows machines are still supported in this release, but in a future release support will be completely removed for these devices.

# R2014a

## Number of local workers no longer limited to 12

You can now run a local cluster of more than 12 workers on your client machine. Unless you adjust the cluster profile, the default maximum size for a local cluster is the same as the number of computational cores on the machine.

## Additional GPU-enabled MATLAB functions: `interp3`, `interpn`, `besselj`, `bessely`

The following functions are new in their support for gpuArrays:

```
besselj
bessely
interp3
interpn
```

For a list of MATLAB functions that support gpuArray, see Run Built-In Functions on a GPU.

## Additional GPU enabled Image Processing Toolbox functions: `bwdist`, `imreconstruct`, `iradon`, `radon`

Image Processing Toolbox™ offers enhanced functionality for some of its functions to perform computations on a GPU. For specific information, see the Image Processing Toolbox release notes. Parallel Computing Toolbox is required to access this functionality.

## Enhancements to GPU-enabled MATLAB functions: `filter` (IIR filters); `pagefun` (additional functions supported); `interp1`, `interp2`, `conv2`, `reshape` (performance improvements)

The following functions are enhanced in their support for gpuArray data:

```
conv2
filter
interp1
interp2
pagefun
rand
randi
randn
reshape
```

Note the following enhancements for some of these functions:

- `filter` now supports IIR filtering.
- pagefun is enhanced to support most element-wise gpuArray functions. Also, these functions are supported: `@ctranspose`, `@fliplr`, `@flipud`, `@mtimes`, `@rot90`, `@transpose`.
- `rand(___,'like',P)` returns a gpuArray of random values of the same underlying class as the gpuArray `P`. This enhancement also applies to randi, randn.

For a list of MATLAB functions that support gpuArray, see Run Built-In Functions on a GPU.

## Duplication of an existing job, containing some or all of its tasks

You can now duplicate job objects, allowing you to resubmit jobs that had finished or failed.

The syntax to duplicate a job is

```
newjob = recreate(oldjob)
```

where `oldjob` is an existing job object. The `newjob` object has all the same tasks and settable properties as `oldjob`, but receives a new `ID`. The old job can be in any state; the new job state is `pending`.

You can also specify which tasks from an existing independent job to include in the new job, based on the task IDs. For example:

```
newjob = recreate(oldjob,'TaskID',[33:48]);
```

For more information, see the recreate reference page.

## More MATLAB functions enhanced for distributed arrays

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

```
eye
ifft
rand
randi
randn
```

Note the following enhancements for some of these functions:

- `ifft` and `randi` are new in support of distributed and codistributed arrays.
- `rand(___,'like',D)` returns a distributed or codistributed array of random values of the same underlying class as the distributed or codistributed array `D`. This enhancement also applies to randi, randn, and eye.

For a list of MATLAB functions that support distributed arrays, see MATLAB Functions on Distributed and Codistributed Arrays.

## GPU MEX Support for Updated MEX

The GPU MEX support for CUDA code now incorporates the latest MATLAB MEX functionality. See Streamlined MEX compiler setup and improved troubleshooting.

## Compatibility Considerations

The latest MEX does not support the `mexopts` files shipped in previous releases. Updated `.xml` files are provided instead. To use the updated MEX functionality and to avoid a warning, replace the `mexopts` file you used in past releases with the appropriate new `.xml` file as described in Set Up for MEX-File Compilation.

## Old Programming Interface Removed

The programming interface characterized by distributed jobs and parallel jobs has been removed. This old interface used functions such as `findResource`, `createParallelJob`, `getAllOutputArguments`, `dfeval`, etc.

## Compatibility Considerations

The functions of the old programming interface now generate errors. You must migrate your code to the interface described in the R2012a release topic “New Programming Interface” on page 7-2.

## `matlabpool` Function Being Removed

The `matlabpool` function is being removed.

## Compatibility Considerations

Calling `matlabpool` continues to work in this release, but now generates a warning. You should instead use parpool to create a parallel pool.

## Removed Support for parallel.cluster.Mpiexec

Support for clusters of type parallel.cluster.Mpiexec has been removed.

## Compatibility Considerations

Any attempt to use parallel.cluster.Mpiexec clusters now generates an error. As an alternative, consider using the generic scheduler interface, parallel.cluster.Generic.

# R2013b

# `parpool`: New command-line interface (replaces `matlabpool`), desktop indicator, and preferences for easier interaction with a parallel pool of MATLAB workers



## Parallel Pool

### Replaces MATLAB Pool

Parallel pool syntax replaces MATLAB pool syntax for executing parallel language constructs such as `parfor`, `spmd`, `Composite`, and `distributed`. The pool is represented in MATLAB by a parallel.Pool object.

The general workflow for these parallel constructs remains the same: When the pool is available, `parfor`, `spmd`, etc., run the same as before. For example:

| MATLAB Pool | Parallel Pool |
|---|---|
| `matlabpool open 4`<br>`parfor ii=1:n`<br>`    X(n) = myFun(n);`<br>`end`<br>`matlabpool close` | `p = parpool(4)`<br>`parfor ii=1:n`<br>`    X(n) = myFun(n);`<br>`end`<br>`delete(p)` |

### Functions for Pool Control

The following functions provide command-line interface for controlling a parallel pool. See the reference pages for more information about each.

| Function | Description |
|---|---|
| parpool | Start a parallel pool |
| gcp | Get current parallel pool or start pool |
| delete | Shut down and delete the parallel pool |
| addAttachedFiles | Attach files to the parallel pool |
| listAutoAttachedFiles | List files automatically attached to the parallel pool |
| updateAttachedFiles | Send updates to files already attached to the parallel pool |

### Asynchronous Function Evaluation on Parallel Pool

You can evaluate functions asynchronously on one or all workers of a parallel pool. Use `parfeval` to evaluate a function on only one worker, or use `parfevalOnAll` to evaluate a function on all workers in the pool.

`parfeval` or `parfevalOnAll` returns an object called a *future*, from which you can get the outputs of the asynchronous function evaluation. You can create an array of futures by calling `parfeval` in a `for`-loop, setting unique parameters for each call in the array.

The following table lists the functions for submitting asynchronous evaluations to a parallel pool, retrieving results, and controlling future objects. See the reference page of each for more details.

| Function | Description |
|---|---|
| parfeval | Evaluate function asynchronously on worker in parallel pool |
| parfevalOnAll | Evaluate function asynchronously on all workers in parallel pool |
| fetchOutputs | Retrieve all output arguments from future |
| fetchNext | Retrieve next available unread future outputs |
| cancel | Cancel queued or running future |
| wait | Wait for future to complete |

For more information on parallel future objects, including their methods and properties, see the parallel.Future reference page.

## Compatibility Considerations

This release continues to support MATLAB pool language usage, but this support might discontinue in future releases. You should update your code as soon as possible to use parallel pool syntax instead.

### New Desktop Pool Indicator

A new icon at the lower-left corner of the desktop indicates the current pool status.

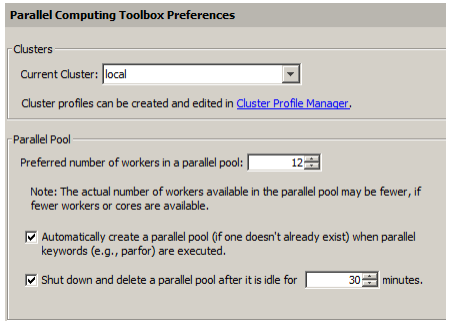
Icon color and tool tips let you know if the pool is busy or ready, how large it is, and when it might time out. You can click the icon to start a pool, stop a pool, or access your parallel preferences.

### New Parallel Preferences

Your MATLAB preferences now include a group of settings for parallel preferences. These settings control general behavior of clusters and parallel pools for your MATLAB session.

You can access your parallel preferences in these ways:

- In the **Environment** section of the **Home** tab, click **Parallel** > **Parallel Preferences**
- Click the desktop pool indicator icon, and select **Parallel preferences**.
- Type `preferences` at the command line, and click the Parallel Computing Toolbox node.




Settings in your parallel preferences control the default cluster choice, and the preferred size, automatic opening, and timeout conditions for parallel pools. For more information about these settings, see Parallel Preferences.

## Automatic start of a parallel pool when executing code that uses `parfor` or `spmd`

You can set your parallel preferences so that a parallel pool automatically starts whenever you execute a language construct that runs on a pool, such as `parfor`, `spmd`, `Composite`, `distributed`, `parfeval`, and `parfevalOnAll`.

## Compatibility Considerations

The default preference setting is to automatically start a pool when a parallel language construct requires it. If you want to make sure a pool does not start automatically, you must change your parallel preference setting. You can also work around this by making sure to explicitly start a parallel pool with parpool before encountering any code that needs a pool.

By default, a parallel pool will shut down *if idle for 30 minutes*. To prevent this, change the setting in your parallel preferences; or the pool indicator tool tip warns of an impending timeout and provides a link to extend it.

## Option to start a parallel pool without using MPI

You now have the option to start a parallel pool on a local or MJS cluster so that the pool does not support running SPMD constructs. This allows the parallel pool to keep running even if one or more workers aborts during `parfor` execution. You explicitly disable SPMD support when starting the parallel pool by setting its `'SpmdEnabled'` property `false` in the call to the `parpool` function. For example:

```
p = parpool('SpmdEnabled',false);
```

## Compatibility Considerations

Running any code (including MathWorks toolbox code) that uses SPMD constructs, on a parallel pool that was created without SPMD support, will generate errors.

## More GPU-enabled MATLAB functions (e.g., `interp2`, `pagefun`) and Image Processing Toolbox functions (e.g., `bwmorph`, `edge`, `imresize`, and `medfilt2`)

Image Processing Toolbox offers enhanced functionality for some of its functions to perform computations on a GPU. For specific information, see the Image Processing Toolbox release notes. Parallel Computing Toolbox is required to access this functionality.

The following functions are new or enhanced in Parallel Computing Toolbox to support gpuArrays:

`flip`

```
interp2
pagefun
```

Note the following for some of these functions:

- `pagefun` allows you to iterate over the pages of a gpuArray, applying `@mtimes` to each page.

  For more information, see the pagefun reference page, or type `help pagefun`.

For complete lists of functions that support gpuArray, see Run Built-In Functions on a GPU.

## More MATLAB functions enabled for distributed arrays: `permute`, `ipermute`, and `sortrows`

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

| | |
|---|---|
| `ipermute` | `zeros` |
| `permute` | `ones` |
| `sortrows` | `nan` |
| | `inf` |
| `cast` | `true` |
| | `false` |

Note the following enhancements for some of these functions:

- `ipermute`, `permute`, and `sortrows` support distributed arrays for the first time in this release.
- `cast` supports the `'like'` option for distributed arrays, applying the underlying class of one array to another.
- `Z = zeros(___,'like',P)` returns a distributed array of zeros of the same complexity as distributed array `P`, and same underlying class as `P` if class is not specified in the function call. The same behavior applies to the other similar constructors in the right-hand column of this table.

For more information on any of these functions, type `help distributed.`*`functionname`*. For example:

```
help distributed.ipermute
```

For complete lists of MATLAB functions that support distributed arrays, see MATLAB Functions on Distributed and Codistributed Arrays.

## Enhancements to MATLAB functions enabled for GPUs, including `ones`, `zeros`

The following functions are enhanced in their support for gpuArray data:

```
zeros                                   eye
ones                                    true
nan                                     false
inf
                                        cast
```

Note the following enhancements for these functions:

- `Z = zeros(___,'like',P)` returns a gpuArray of zeros of the same complexity as gpuArray `P`, and same underlying class as `P` if class is not specified. The same behavior applies to the other constructor functions listed in this table.
- `cast` also supports the `'like'` option for gpuArray input, applying the underlying class of one array to another.

For more information on any of these functions, type `help gpuArray.`*`functionname`*. For example:

```
help gpuArray.cast
```

For complete lists of MATLAB functions that support gpuArray, see Run Built-In Functions on a GPU.

## `gputimeit` Function to Time GPU Computations

`gputimeit` is a new function to measure the time to run a function on a GPU. It is similar to the MATLAB function timeit, but ensures accurate time measurement on the GPU. For more information and examples, see the gputimeit reference page.

## New GPU Random Number Generator NormalTransform Option: Box-Muller

When generating random numbers on a GPU, there is a new option for `'NormalTransform'` called `'BoxMuller'`. The Box-Muller transform allows faster generation of normally distributed random numbers on the GPU.

This new option is the default `'NormalTransform'` setting when using the `Philox4x32-10` or `Threefry4x64-20` generator. The following commands, therefore, use `'BoxMuller'` for `'NormalTransform'`:

```
parallel.gpu.rng(0,'Philox4x32-10')
parallel.gpu.rng(0,'Threefry4x64-20')
```

---

**Note** The `'BoxMuller'` option is not supported for the `CombRecursive` (`mrg32k3a`) generator

---

## Compatibility Considerations

In previous releases, the default `'NormalTransform'` setting when using the `Philox4x32-10` or `Threefry4x64-20` generator on a GPU was `'Inversion'`. If you used either of these generators with the default `'NormalTranform'` and you want to continue with the same behavior, you must explicitly set the `'NormalTransform'` with either of these commands:

```
stream = parallel.gpu.RandStream('Philox4x32-10','NormalTransform','Inversion')
parallel.gpu.RandStream.setGlobalStream(stream)
```

```
stream = parallel.gpu.RandStream('Threefry4x64-20','NormalTransform','Inversion')
parallel.gpu.RandStream.setGlobalStream(stream)
```

## Upgraded MPICH2 Version

The parallel computing products are now shipping MPICH2 version 1.4.1p1 on all platforms.

## Compatibility Considerations

If you use your own MPI builds, you might need to create new builds compatible with this latest version, as described in Use Different MPI Builds on UNIX Systems.

## Discontinued Support for GPU Devices of Compute Capability 1.3

In a future release, support for GPU devices of compute capability 1.3 will be removed. At that time, a minimum compute capability of 2.0 will be required.

## Compatibility Considerations

In R2013b, any use of `gpuDevice` to select a GPU with compute capability 1.3, generates a warning. The device is still supported in this release, but in a future release support will be completely removed for these 1.3 devices.

## Discontinued Support for parallel.cluster.Mpiexec

Support for clusters of type parallel.cluster.Mpiexec is being discontinued.

## Compatibility Considerations

In R2013b, any use of parallel.cluster.Mpiexec clusters generates a warning. In a future release, support will be completely removed.

# R2013a

Version: 6.2

New Features

Bug Fixes

## GPU-enabled functions in Image Processing Toolbox and Phased Array System Toolbox

More toolboxes offer enhanced functionality for some of their functions to perform computations on a GPU. For specific information about these other toolboxes, see their respective release notes. Parallel Computing Toolbox is required to access this functionality.

## More MATLAB functions enabled for use with GPUs, including `interp1` and `ismember`

The following functions are enhanced to support gpuArray data:

```
interp1                 ismember
isfloat                 isnumeric
isinteger
```

Note the following for some of these functions:

- `interp1` supports only the linear and nearest interpolation methods.
- `isfloat`, `isinteger`, and `isnumeric` now return results based on `classUnderlying` of the gpuArray.
- `ismember` does not support the `'rows'` or `'legacy'` option for gpuArray input.

For complete lists of functions that support gpuArray, see Built-In Functions That Support gpuArray.

## Enhancements to MATLAB functions enabled for GPUs, including `arrayfun`, `svd`, and `mldivide` (\)

The following functions are enhanced in their support for gpuArray data:

```
arrayfun                mrdivide
bsxfun                  svd
mldivide
```

Note the following enhancements for some of these functions:

- `arrayfun` and `bsxfun` support indexing and accessing variables of outer functions from within nested functions.
- `arrayfun` supports singleton expansion of all arguments for all operations. For more information, see the arrayfun reference page.
- `mldivide` and `mrdivide` support all rectangular arrays.
- `svd` can perform economy factorizations.

For complete lists of MATLAB functions that support gpuArray, see Built-In Functions That Support gpuArray.

## Ability to launch CUDA code and manipulate data contained in GPU arrays from MEX-functions

You can now compile MEX-files that contain CUDA code, to create functions that support gpuArray input and output. This functionality is supported only on 64-bit platforms (win64, glnxa64, maci64). For more information and examples, see Execute MEX-Functions Containing CUDA Code.

For a list of C functions supporting this capability, see the group of C Functions in GPU Computing.

## Automatic detection and transfer of files required for execution in both batch and interactive workflows

Parallel Computing Toolbox can now automatically attach files to a job so that workers have the necessary code files for evaluating tasks. When you set a job object's `AutoAttachFiles` to true, an analysis determines what files on the client machine are necessary for the evaluation of your job, and those files are automatically attached to the job and sent to the worker machines.

You can set the `AutoAttachFiles` property in the Cluster Profile Manager, or at the command-line. To get a listing of the files that are automatically attached, use the listAutoAttachedFiles method on a job or task object.

For more information, see Pass Data to and from Worker Sessions.

If the `AutoAttachFiles` property in the cluster profile for the MATLAB pool is set to `true`, MATLAB performs an analysis on spmd blocks and parfor-loops to determine what

code files are necessary for their execution, then automatically attaches those files to the MATLAB pool job so that the code is available to the workers.

When you use the MATLAB editor to update files on the client that are attached to a matlabpool, those updates are automatically propagated to the workers in the pool.

## More MATLAB functions enabled for distributed arrays

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

```
cumprod                 qr
cumsum                  prod
eig
```

Note the following enhancements for some of these functions:

- `eig` supports generalized eigenvalues for symmetric matrices.
- `qr` supports column pivoting.
- `cumprod`, `cumsum`, and `prod` now support all integer data types; and `prod` accepts the optional `'native'` or `'double'` argument.

# R2012b

## More MATLAB functions enabled for GPUs, including `convn`, `cov`, and `normest`



### gpuArray Support

The following functions are enhanced to support gpuArray data, or are expanded in their support:

```
bitget                  cov                     normest
bitset                  issparse                pow2
cond                    mpower                  var
convn                   nnz
```

The following functions are not methods of the gpuArray class, but they now work with gpuArray data:

```
blkdiag                 ismatrix                isvector
cross                   isrow                   std
iscolumn
```

For complete lists of functions that support gpuArray, see Built-In Functions That Support gpuArray.

### MATLAB Code on the GPU

`bsxfun` now supports the same subset of the language on a GPU that `arrayfun` does.

GPU support is extended to include the following MATLAB code in functions called by `arrayfun` and `bsxfun` to run on the GPU:

```
bitget
bitset
pow2
```

## GPU-enabled functions in Neural Network Toolbox, Phased Array System Toolbox, and Signal Processing Toolbox

A number of other toolboxes now support enhanced functionality for some of their functions to perform computations on a GPU. For specific information about these other

toolboxes, see their respective release notes. Parallel Computing Toolbox is required to access this functionality.

## Performance improvements to GPU-enabled MATLAB functions and random number generation

The performance of some MATLAB functions and random number generation on GPU devices is improved in this release.

You now have a choice of three random generators on the GPU: the combined multiplicative recursive MRG32K3A, the Philox4x32-10, and the Threefry4x64-20. For information on these generators and how to select them, see Control the Random Stream for gpuArray. For information about generating random numbers on a GPU, and a comparison between GPU and CPU generation, see Control Random Number Streams.

## Automatic detection and selection of specific GPUs on a cluster node when multiple GPUs are available on the node

When multiple workers run on a single compute node with multiple GPU devices, the devices are automatically divided up among the workers. If there are more workers than GPU devices on the node, multiple workers share the same GPU device. If you put a GPU device in `'exclusive'` mode, only one worker uses that device. As in previous releases, you can change the device used by any particular worker with the `gpuDevice` function.

## More MATLAB functions enabled for distributed arrays, including `sparse` constructor, `bsxfun`, and `repmat`

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

```
atan2d                  mrdivide                struct2cell
bsxfun                  repmat                  typecast
cell2struct             sparse
```

Note the following enhancements for some of these functions:

- `cell2struct`, `struct2cell`, and `typecast` now support 2DBC in addition to 1-D distribution.

- `mrdivide` is now fully supported, and is no longer limited to accepting only scalars for its second argument.
- `sparse` is now fully supported for all distribution types.

This release also offers improved performance of fft functions for long vectors as distributed arrays.

## Detection of MATLAB Distributed Computing Server clusters that are available for connection from user desktops through Profile Manager

You can let MATLAB discover clusters for you. Use either of the following techniques to discover those clusters which are available for you to use:

- On the **Home** tab in the **Environment** section, click **Parallel > Discover Clusters**.
- In the Cluster Profile Manager, click **Discover Clusters**.

For more information, see Discover Clusters.

## gpuArray Class Name

The object type formerly known as a GPUArray has been renamed to gpuArray. The corresponding class name has been changed from parallel.gpu.GPUArray to the shorter gpuArray. The name of the function `gpuArray` remains unchanged.

## Compatibility Considerations

You cannot load gpuArray objects from files that were saved in previous versions.

Code that uses the old class name must be updated to use the shorter new name. For example, the functions for directly generating gpuArrays on the GPU:

| Previous version form | New version form |
|---|---|
| `parallel.gpu.GPUArray.rand`<br>`parallel.gpu.GPUArray.ones`<br>etc. | `gpuArray.rand`<br>`gpuArray.ones`<br>etc. |

## Diary Output Now Available During Running Task

Diary output from tasks (including those of batch jobs) can now be obtained while the task is still running. The diary is text output that would normally be sent to the

Command Window. Now this text is appended to the task's `Diary` property as the text is generated, rather than waiting until the task is complete. You can read this property at any time. Diary information is accumulated only if the job's `CaptureDiary` property value is `true`. (**Note**: This feature is not yet available for SOA jobs on HPC Server clusters.)

# R2012a

## New Programming Interface

This release provides a new programming interface for accessing clusters, jobs, and tasks.

### General Concepts and Phrases

This table maps some of the concepts and phrases from the old interface to the new.

| Previous Interface | New Interface in R2012a |
|---|---|
| MathWorks job manager | MATLAB job scheduler (MJS) |
| Third-party or local scheduler | Common job scheduler (CJS) |
| Configuration | Profile |
| Scheduler | Cluster |

The following code examples compare programming the old and new interfaces, showing some of the most common commands and properties. Note that most differences involve creating a cluster object instead of a scheduler object, and some of the property and method names on the job. After the example are tables listing some details of these new objects.

| Previous Interface | New Interface |
|---|---|
| `sched = findResource('scheduler',...`<br>`'Configuration','local');`<br>`j = createJob(sched,...`<br>`'PathDependencies',{'/share/app/'},...`<br>`'FileDependencies',{'funa.m','funb.m'};`<br>`createTask(j,@myfun,1,{3,4});`<br>`submit(j);`<br>`waitForState(j);`<br>`results = j.getAllOutputArguments;` | `clust = parcluster('local');`<br><br>`j = createJob(clust,...`<br>`'AdditionalPaths',{'/share/app/'}),...`<br>`'AttachedFiles,{'funa.m','funb.m'};`<br>`createTask(j,@myfun,1,{3,4});`<br>`submit(j);`<br>`wait(j);`<br>`results = j.fetchOutputs;` |

### Objects

These tables compare objects in the previous and new interfaces.

| Previous Scheduler Objects | New Cluster Objects |
|---|---|
| `ccsscheduler` | `parallel.cluster.HPCServer` |
| `genericscheduler` | `parallel.cluster.Generic` |
| `jobmanager` | `parallel.cluster.MJS` |

| Previous Scheduler Objects | New Cluster Objects |
|---|---|
| `localscheduler` | `parallel.cluster.Local` |
| `lsfscheduler` | `parallel.cluster.LSF` |
| `mpiexec` | `parallel.cluster.Mpiexec` |
| `pbsproscheduler` | `parallel.cluster.PBSPro` |
| `torquescheduler` | `parallel.cluster.Torque` |

For information on each of the cluster objects, see the parallel.Cluster reference page.

| Previous Job Objects | New Job Objects |
|---|---|
| `job` (distributed job) | `parallel.job.MJSIndependentJob` |
| `matlabpooljob` | `parallel.job.MJSCommunicatingJob` where `('Type' = 'Pool')` |
| `paralleljob` | `parallel.job.MJSCommunicatingJob` where `('Type' = 'SPMD')` |
| `simplejob` | `parallel.job.CJSIndependentJob` |
| `simplematlabpooljob` | `parallel.job.CJSCommunicatingJob ('Type' = 'Pool')` |
| `simpleparalleljob` | `parallel.job.CJSCommunicatingJob ('Type' = 'SPMD')` |

For information on each of the job objects, see the parallel.Job reference page.

| Previous Task Objects | New Task Objects |
|---|---|
| `task` | `parallel.task.MJSTask` |
| `simpletask` | `parallel.task.CJSTask` |

For information on each of the task objects, see the parallel.Task reference page.

| Previous Worker Object | New Worker Objects |
|---|---|
| `worker` | `parallel.cluster.MJSWorker`, `parallel.cluster.CJSWorker` |

For information on each of the worker objects, see the parallel.Worker reference page.

### Functions and Methods

This table compares some functions and methods of the old interface to those of the new. Many functions do not have a name change in the new interface, and are not listed here. Not all functions are available for all cluster types.

| Previous Name | New Name |
|---|---|
| `findResource` | parcluster |
| `createJob` | createJob (no change) |
| `createParallelJob` | createCommunicatingJob (where `'Type' = 'SPMD'`) |
| `createMatlabPoolJob` | createCommunicatingJob (where `'Type' = 'Pool'`) |
| `destroy` | delete |
| `clearLocalPassword` | logout |
| `createTask` | createTask (no change) |
| `getAllOutputArguments` | fetchOutputs |
| `getJobSchedulerData` | getJobClusterData |
| `setJobSchedulerData` | setJobClusterData |
| `getCurrentJobmanager` | getCurrentCluster |
| `getFileDependencyDir` | getAttachedFilesFolder |

### Properties

In addition to a few new properties on the objects in the new interface, some other properties have new names when using the new interface, according to the following tables.

| Previous Scheduler Properties | New Cluster Properties |
|---|---|
| `DataLocation` | `JobStorageLocation` |
| `Configuration` | `Profile` |
| `HostAddress` | `AllHostAddresses` |
| `Computer` | `ComputerType` |
| `ClusterOsType` | `OperatingSystem` |

| Previous Scheduler Properties | New Cluster Properties |
|---|---|
| IsUsingSecureCommunication | HasSecureCommunication |
| SchedulerHostname, MasterName, Hostname | Host |
| ParallelSubmissionWrapperScript | CommunicatingJobWrapper |
| ClusterSize | NumWorkers |
| NumberOfBusyWorkers | NumBusyWorkers |
| NumberOfIdleWorkers | NumIdleWorkers |
| SubmitFcn | IndependentSubmitFcn |
| ParallelSubmitFcn | CommunicatingSubmitFcn |
| DestroyJobFcn | DeleteJobFcn |
| DestroyTaskFcn | DeleteTaskFcn |

| Previous Job Properties | New Job Properties |
|---|---|
| FileDependencies | AttachedFiles |
| PathDependencies | AdditionalPaths |
| MinimumNumberOfWorkers, MaximumNumberOfWorkers | NumWorkersRange |

| Previous Task Properties | New Task Properties |
|---|---|
| MaximumNumberOfRetries | MaximumRetries |

### Getting Help

In addition to these changes, there are some new properties and methods, while some old properties are not used in the new interface. For a list of the methods and properties available for clusters, jobs, and tasks, use the following commands for help on each class type:

```
help parallel.Cluster
help parallel.Job
help parallel.Task

help parallel.job.CJSIndependentJob
```

```
help parallel.job.CJSCommunicatingJob
help parallel.task.CJSTask

help parallel.job.MJSIndependentJob
help parallel.job.MJSCommunicatingJob
help parallel.task.MJSTask
```

There might be slight changes in the supported format for properties whose names are still the same. To get help on an individual method or property, the general form of the command is:

```
help parallel.obj-type.method-or-property-name
```

You might need to specify the subclass in some cases, for example, where different cluster types use properties differently. The following examples display the help for specified methods and properties of various object types:

```
help parallel.cluster.LSF.JobStorageLocation
help parallel.Job.fetchOutputs
help parallel.job.MJSIndependentJob.FinishedFcn
help parallel.Task.delete
```

## Compatibility Considerations

### Jobs

*This release still supports the old form of interface*, however, the old interface and the new interface are not compatible in the same job. For example, a job manager scheduler object that you create with `findResource` requires that you use only the old interface methods and properties, and cannot be used for creating a communicating job (`createCommunicatingJob`). A cluster that was defined with `parcluster` must use the new interface, and cannot be used to create a parallel job (`createParallelJob`).

Graphical interfaces provide access only to the new interface. The Configurations Manager is no longer available and is replaced by the Cluster Profile Manager. Actions that use profiles automatically convert and upgrade your configurations to profiles. Therefore, if you already have a group of configurations, the first time you open the Cluster Profile Manager, it should already be populated with your converted profiles. Furthermore, you can specify cluster profiles when using the old interface in places where configurations are expected.

One job manager (or MJS) can accommodate jobs of both the old and new interface at the same time.

### Creating and Finding Jobs and Tasks

In the old interface, to create or find jobs or tasks, you could specify property name and value pairs in structures or cell arrays, and then provide the structure or cell array as an input argument to the function you used. In the new interface, property names and values must be pairs of separate arguments, with the property name as a string expression and its value of the appropriate type. This applies to the functions `createJob`, `createCommunicatingJob`, `createTask`, `findJob`, and `findTask`.

### Batch

Jobs created by the `batch` command use the new interface, even if you specify configurations or properties using the old interface. For example, the following code generates two identical jobs of the new interface, even though job `j2` is defined with an old interface property. The script `randScript` contains just the one line of code, `R = rand(3)`, and the default profile is `local`.

```
j1 = batch('randScript','AdditionalPaths','c:\temp');
j1.wait;
R1 = j1.load('R');
```

or

```
j2 = batch('randScript','PathDependencies','c:\temp');
j2.wait;
R2 = j2.load('R');
```

```
 whos
  Name      Size      Bytes  Class

  R1        1x1         248  struct
  R2        1x1         248  struct
  j1        1x1         112  parallel.job.CJSIndependentJob
  j2        1x1         112  parallel.job.CJSIndependentJob
```

#### Communicating Job Wrapper

In the old interface, for a parallel job in an LSF, TORQUE, or PBS Pro scheduler, you would call the scheduler's `setupForParallelExecution` method with the necessary arguments so that the toolbox could automatically set the object's `ClusterOSType` and `ParallelSubmissionWrapperScript` properties, thus determining which wrapper was used. In the new interface, with a communicating job you only have to set the LSF, Torque, or PBSPro cluster object's `OperatingSystem` and `CommunicatingJobWrapper`

properties, from which the toolbox calculates which wrapper to use. For more information about these properties and their possible values, in MATLAB type

```
help parallel.cluster.LSF.CommunicatingJobWrapper
```

You can change the `LSF` to `PBSPro` or `Torque` in this command, as appropriate.

### Enhanced Example Scripts

An updated set of example scripts is available in the product for using a generic scheduler with the new programming interface. These currently supported scripts are provided in the folder:

*`matlabroot`*`/toolbox/distcomp/examples/integration`

For more information on these scripts and their updates, see the `README` file provided in each subfolder, or see Supplied Submit and Decode Functions.

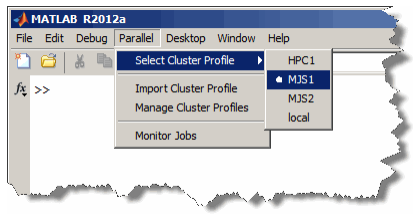
## Compatibility Considerations

Scripts that use the old programming interface are provided in the folder *`matlabroot`*`/toolbox/distcomp/examples/integration/old`. The scripts that resided in this folder in previous releases are no longer available. The scripts currently in this folder might be removed in future releases.

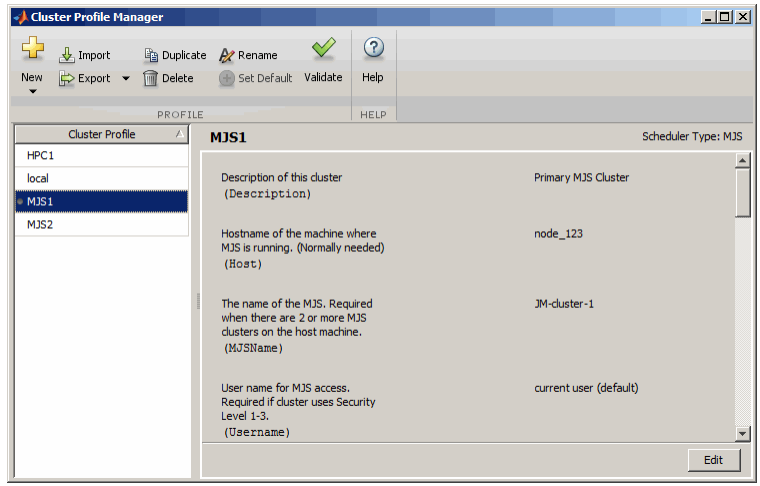
## Cluster Profiles

### New Cluster Profile Manager

Cluster profiles replace parallel configurations for defining settings for clusters and their jobs. You can access profiles and the Cluster Profile Manager from the desktop **Parallel** menu. From this menu you can select or import profiles. To choose a profile as the default, select the desktop menu **Parallel > Select Cluster Profile**. The current default profile is indicated with a bold dot.



The Cluster Profile Manager lets you create, edit, validate, import, and export profiles, among other actions. To open the Cluster Profile Manager, select **Parallel > Manage Cluster Profiles**.




For more information about cluster profiles and the Cluster Profile Manager, see Cluster Profiles.

### Programming with Profiles

These commands provide access to profiles and the ability to create cluster objects.

| Function | Description |
|---|---|
| `p = parallel.clusterProfiles` | List of your profiles |
| parallel.defaultClusterProfile | Specifies which profile to use by default |
| `c = parcluster('clustername')` | Creates cluster object, `c`, for accessing parallel compute resources |
| `c.saveProfile` | Saves changes of cluster property values to its current profile |
| `c.saveAsProfile` | Saves cluster properties to a profile of the specified name, and sets that as the current profile for this cluster |
| parallel.importProfile | Import profiles from specified `.settings` file |

| Function | Description |
|---|---|
| parallel.exportProfile | Export profiles to specified `.settings` file |

### Profiles in Compiled Applications

Because compiled applications include the current profiles of the user who compiles, in most cases the application has the profiles it needs. When other profiles are needed, a compiled application can also import profiles that have been previously exported to a `.settings` file. The new `ParallelProfile` key supports exported parallel configuration `.mat` files and exported cluster profile `.settings` files; but this might change in a future release. For more information, see Export Profiles for MATLAB Compiler.

## Compatibility Considerations

In past releases, when a compiled application imported a parallel configuration, that configuration would overwrite a configuration of the same if it existed. In this release, imported profiles are renamed if profiles already exist with the same name; so the same profile might have a different name in the compiled application than it does in the exported profile file.

## Enhanced GPU Support

### GPUArray Support

The following functions are added to those that support GPUArray data, or are enhanced in their support for this release:

```
beta                ifft                mldivide
betaln              ifft2               min
bsxfun              ifftn               mrdivide
circshift           ind2sub             norm
det                 int2str             num2str
eig                 inv                 permute
fft                 ipermute            qr
fft2                isequaln            shiftdim
fftn                issorted            sprintf
fprintf             mat2str             sub2ind
full                max
```

Note the following enhancements and restrictions to some of these functions:

- GPUArray usage now supports all data types supported by MATLAB, except `int64` and `uint64`.
- For the list of functions that `bsxfun` supports, see the bsxfun reference page.
- The full range of syntax is now supported for `fft`, `fft2`, `fftn`, `ifft`, `ifft2`, and `ifftn`.
- `eig` now supports all matrices, symmetric or not.
- `issorted` supports only vectors, not matrices.
- `max` and `min` can now return two output arguments, including an index vector.
- `mldivide` supports complex arrays. It also supports overdetermined matrices (with more rows than columns), with no constraints on the second input.
- `mrdivide` supports underdetermined matrices (more columns than rows) as the second input argument.
- `norm` now supports the form `norm(X,2)`, where `X` is a matrix.

The following functions are not methods of the GPUArray class, but they do work with GPUArray data:

```
angle                ifftshift            rank
fliplr               kron                 squeeze
flipud               mean                 rot90
flipdim              perms                trace
fftshift
```

### Reset or Deselect GPU Device

Resetting a GPU device clears your GPUArray and CUDAKernel data from the device. There are two ways to reset a GPU device, while still keeping it as the currently selected device. You can use the reset function, or you can use `gpuDevice(idx)` with the current device's index for `idx`. For example, use `reset`:

```
idx = 1
g = gpuDevice(idx)
reset(g)
```

Alternatively, call `gpuDevice` again with the same index argument:

```
idx = 1
```

```
g = gpuDevice(idx)
.
.
.
g = gpuDevice(idx)  % Resets GPU device, clears data
```

To deselect the current device, use `gpuDevice([ ])` with an empty argument (as opposed to no argument). This clears the GPU of all arrays and kernels, and invalidates variables in the workspace that point to such data.

### Asynchronous GPU Calculations and Wait

All GPU calculations now run asynchronously with MATLAB. That is, when you initiate a calculation on the GPU, MATLAB continues execution while the GPU runs its calculations at the same time. The wait command now accommodates a GPU device, so that you can synchronize MATLAB and the GPU. The form of the command is

```
wait(gpudev)
```

where `gpudev` is the object representing the GPU device to wait for. At this command, MATLAB waits until all current calculations complete on the specified device.

## Compatibility Considerations

In previous releases, MATLAB and the GPU were synchronous, so that any calls to the GPU had to complete before MATLAB proceeded to the next command. This is no longer the case. Now MATLAB continues while the GPU is running. The `wait` command lets you time GPU code execution.

### Verify GPUArray or CUDAKernel Exists on the Device

The new function existsOnGPU lets you verify that a GPUArray or CUDAKernel exists on the current GPU device, and that its data is accessible from MATLAB. It is possible to reset the GPU device, so that a GPUArray or CUDAKernel object variable still exists in your MATLAB workspace even though it is no longer available on the GPU. For example, you can reset a GPU device using the command `gpuDevice(index)` or `reset(dev)`:

```
index = 1;
g = gpuDevice(index);
R = parallel.gpu.GPUArray.rand(4,4)
    0.5465    0.3000    0.4067    0.6110
    0.9024    0.8965    0.6635    0.7709
```

```
    0.8632    0.7481    0.9901    0.0420
    0.2307    0.7008    0.7516    0.5059

existsOnGPU(R)
     1

reset(g);  % Resets GPU device
existsOnGPU(R)
     0

R  % View GPUArray contents
Data no longer exists on the GPU.
```

Any attempt to use the data from `R` generates an error.

### MATLAB Code on the GPU

GPU support is extended to include the following MATLAB code in functions called by arrayfun to run on the GPU:

| | | |
|---|---|---|
| `and` | `intmin` | `power` |
| `beta` | `ldivide` | `rdivide` |
| `betaln` | `le` | `realmax` |
| `eq` | `lt` | `realmin` |
| `ge` | `minus` | `times` |
| `gt` | `ne` | `uint8` |
| `int8` | `not` | `uint16` |
| `int16` | `or` | |
| `intmax` | `plus` | |

Note the following enhancements and restrictions to some of these functions:

- `plus`, `minus`, `ldivide`, `rdivide`, `power`, `times`, and other arithmetic, comparison or logical operator functions were previously supported only when called with their operator symbol. Now they are supported in their functional form, so can be used as direct argument inputs to `arrayfun` and `bsxfun`.
- `arrayfun` and `bsxfun` on the GPU now support the integer data types `int8`, `uint8`, `int16`, and `uint16`.

The code in your function can now call any functions defined in your function file or on the search path. You are no longer restricted to calling only those supported functions listed in the table of Supported MATLAB Code.

### Set CUDA Kernel Constant Memory

The new `setConstantMemory` method on the CUDAKernel object lets you set kernel constant memory from MATLAB. For more information, see the setConstantMemory reference page.

### Latest NVIDIA CUDA Device Driver

This version of Parallel Computing Toolbox GPU functionality supports only the latest NVIDIA CUDA device driver.

## Compatibility Considerations

Earlier versions of the toolbox supported earlier versions of CUDA device driver. Always make sure you have the latest CUDA device driver.

## Enhanced Distributed Array Support

### Newly Supported Functions

The following functions now support distributed arrays with all forms of codistributor (1-D and 2DBC), or are enhanced in their support for this release:

`isequaln`

## Random Number Generation on Workers

MATLAB worker sessions now generate random number values using the combined multiplicative recursive generator (mrg32k3a) by default.

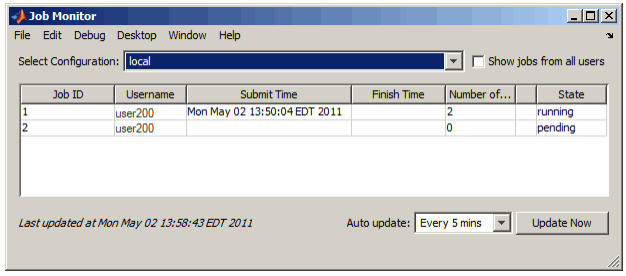
## Compatibility Considerations

In past releases, MATLAB workers used the same default generator as a MATLAB client session. This is no longer the case.

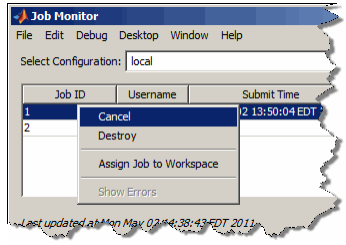
# R2011b

## New Job Monitor

The Job Monitor is a tool that lets you track the jobs you have submitted to a cluster. It displays the jobs for the scheduler determined by your selection of a parallel configuration. Open the Job Monitor from the MATLAB desktop by selecting **Parallel** > **Job Monitor**.



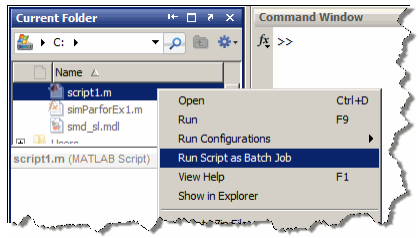

Right-click a job in the list to select a command from the context menu for that job:




For more information about the Job Monitor and its capabilities, see Job Monitor.

## Run Scripts as Batch Jobs from the Current Folder Browser

From the Current Folder browser, you can run a MATLAB script as a batch job by browsing to the file's folder, right-clicking the file, and selecting **Run Script as Batch Job**. The batch job runs on the cluster identified by the current default parallel configuration. The following figure shows the menu option to run the script from the file `script1.m`:




## Number of Local Workers Increased to Twelve

You can now run up to 12 local workers on your MATLAB client machine. If you do not specify the number of local workers in a command or configuration, the default number of local workers is determined by the value of the local scheduler's ClusterSize property, which by default equals the number of computational cores on the client machine.

## Enhanced GPU Support

### Latest NVIDIA CUDA Device Driver

This version of Parallel Computing Toolbox GPU functionality supports only the latest NVIDIA CUDA device driver.

## Compatibility Considerations

Earlier versions of the toolbox supported earlier versions of CUDA device driver. Always make sure you have the latest CUDA device driver.

### Deployment of GPU Applications

MATLAB Compiler™ generated standalone executables and components now support applications that use the GPU.

### Random Number Generation

You can now directly create arrays of random numbers on the GPU using these new static methods for GPUArray objects:

```
parallel.gpu.GPUArray.rand
parallel.gpu.GPUArray.randi
parallel.gpu.GPUArray.randn
```

The following functions set the GPU random number generator seed and stream:

```
parallel.gpu.rng
parallel.gpu.RandStream
```

Also, `arrayfun` called with GPUArray data now supports `rand`, `randi`, and `randn`. For more information about using `arrayfun` to generate random matrices on the GPU, see Generating Random Numbers on the GPU.

### GPUArray Support

The following functions now support GPUArray data:

```
chol                isnan               norm
diff                lu                  not
eig                 max                 repmat
find                min                 sort
isfinite            mldivide            svd
isinf
```

`mldivide` supports complex arrays. It also supports overdetermined matrices (with more rows than columns) when the second input argument is a column vector (has only one column).

`eig` supports only symmetric matrices.

`max` and `min` return only one output argument; they do not return an index vector.

The following functions are not methods of the GPUArray class, but they do work with GPUArray data:

```
angle               flipdim             mean
beta                fftshift            perms
betaln              ifftshift           squeeze
fliplr              kron                rot90
flipud
```

The default display of GPUArray variables now shows the array contents. In previous releases, the display showed some of the object properties, but not the contents. For example, the new enhanced display looks like this:

```
M = gpuArray(magic(3))
M =
     8     1     6
```

```
     3     5     7
     4     9     2
```

To see that `M` is a GPUArray, use the `whos` or `class` function.

### MATLAB Code on the GPU

GPU support is extended to include the following MATLAB code in functions called by arrayfun to run on the GPU:

```
rand
randi
randn
xor
```

Also, the handle passed to `arrayfun` can reference a simple function, a subfunction, a nested function, or an anonymous function. The function passed to `arrayfun` can call any number of its subfunctions. The only restriction is that when running on the GPU, nested and anonymous functions do not have access to variables in the parent function workspace. For more information on function structure and relationships, see Types of Functions.

## Enhanced Distributed Array Support

### Newly Supported Functions

The following functions are enhanced to support distributed arrays, supporting all forms of codistributor (1-D and 2DBC):

```
inv
meshgrid
ndgrid
sort
```

The following functions can now directly construct codistributed arrays:

```
codistributed.linspace(m, n, ..., codist)
codistributed.logspace(m, n, ..., codist)
```

## Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Parallel Computing Toolbox.

## Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a try/catch statement and performs an action based on a specific error identifier.

For example, the `'distcomp:old:ID'` identifier has changed to `'parallel:similar:ID'`. If your code checks for `'distcomp:old:ID'`, you must update it to check for `'parallel:similar:ID'` instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following commands just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

**Tip** Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so that it runs warning-free.

## Task Error Properties Updated

If there is an error during task execution, the task `Error` property now contains the non-empty `MException` object that is thrown. If there was no error during task execution, the `Error` property is empty.

The identifier and message properties of this object are now the same as the task's `ErrorIdentifier` and `ErrorMessage` properties, respectively. For more information about these properties, see the Error, ErrorIdentifier, and ErrorMessage reference pages.

## Compatibility Considerations

In past releases, when there was no error, the `Error` property contained an MException object with empty data fields, generated by `MException('', '')`. Now to determine if a task is error-free, you can query the `Error` property itself to see if it is empty:

```
didTaskError = ~isempty(t.Error)
```

where `t` is the task object you are examining.

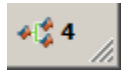
# R2011a

## Deployment of Local Workers

MATLAB Compiler generated standalone executables and libraries from parallel applications can now launch up to eight local workers without requiring MATLAB Distributed Computing Server software.

## New Desktop Indicator for MATLAB Pool Status

When you first open a MATLAB pool from your desktop session, an indicator appears in the lower-right corner of the desktop to show that this desktop session is connected to an open pool. The number indicates how many workers are in the pool.




When you close the pool, the indicator remains displayed and shows a value of 0.

## Enhanced GPU Support

### Static Methods to Create GPUArray

The following new static methods directly create GPUArray objects:

```
parallel.gpu.GPUArray.linspace
parallel.gpu.GPUArray.logspace
```

### GPUArray Support

The following functions are enhanced to support GPUArray data:

```
cat
colon
conv
conv2
cumsum
cumprod
eps
filter
filter2
horzcat
meshgrid
ndgrid
plot
```

```
subsasgn
subsindex
subsref
vertcat
```

and all the plotting related functions.

### GPUArray Indexing

Because GPUArray now supports `subsasgn` and `subsref`, you can index into a GPUArray for assigning and reading individual elements.

For example, create a GPUArray and assign the value of an element:

```
n = 1000;
D = parallel.gpu.GPUArray.eye(n);
D(1,n) = pi
```

Create a GPUArray and read the value of an element back into the MATLAB workspace:

```
m = 500;
D = parallel.gpu.GPUArray.eye(m);
one = gather(D(m,m))
```

### MATLAB Code on the GPU

GPU support is extended to include the following MATLAB code in functions called by `arrayfun` to run on the GPU:

```
&, |, ~, &&, ||,
while, if, else, elseif, for, return, break, continue, eps
```

You can now call `eps` with string inputs, so your MATLAB code running on the GPU can include `eps('single')` and `eps('double')`.

### NVIDIA CUDA Driver 3.2 Support

This version of Parallel Computing Toolbox GPU functionality supports only NVIDIA CUDA device driver 3.2.

## Compatibility Considerations

Earlier versions of the toolbox supported earlier versions of CUDA device driver. If you have an older driver, you must upgrade to CUDA device driver version 3.2.

## Distributed Array Support

### Newly Supported Functions

The following functions are enhanced to support distributed arrays, supporting all forms of codistributor (1-D and 2DBC):

```
arrayfun
cat
reshape
```

### Enhanced mtimes Support

The mtimes function now supports distributed arrays that use a 2-D block-cyclic (2DBC) distribution scheme, and distributed arrays that use 1-D distribution with a distribution dimension greater than 2. Previously, `mtimes` supported only 1-D distribution with a distribution dimension of 1 or 2.

The `mtimes` function now returns a distributed array when only one of its inputs is distributed, similar to its behavior for two distributed inputs.

## Compatibility Considerations

In previous releases, `mtimes` returned a replicated array when one input was distributed and the other input was replicated. Now it returns a distributed array.

## Enhanced parfor Support

### Nested for-Loops Inside parfor

You can now create nested `for`-loops inside a `parfor`-loop, and you can use both the `parfor`-loop and `for`-loop variables directly as indices for the sliced array inside the nested loop. See Nested Loops.

## Enhanced Support for Microsoft Windows HPC Server

### Support for 32-Bit Clients

The parallel computing products now support Microsoft Windows HPC Server on 32-bit Windows clients.

### Search for Cluster Head Nodes Using Active Directory

The `findResource` function can search Active Directory to identify your cluster head node. For more information, see the findResource reference page.

## Enhanced Admin Center Support

You can now start and stop mdce services on remote hosts from Admin Center. For more information, see Start mdce Service.

## New Remote Cluster Access Object

New functionality is available that lets you mirror job data from a remote cluster to a local data location. This supports the generic scheduler interface when making remote submissions to a scheduler or when using a nonshared file system. For more information, see the RemoteClusterAccess object reference page.

# R2010b

## GPU Computing

This release provides the ability to perform calculations on a graphics processing unit (GPU). Features include the ability to:

- Use a GPU array interface with several MATLAB built-in functions so that they automatically execute with single- or double-precision on the GPU — functions including `mldivide`, `mtimes`, `fft`, etc.
- Create kernels from your MATLAB function files for execution on a GPU
- Create kernels from your CU and PTX files for execution on a GPU
- Transfer data to/from a GPU and represent it in MATLAB with GPUArray objects
- Identify and select which one of multiple GPUs to use for code execution

For more information on all of these capabilities and the requirements to use these features, see GPU Computing.

## Job Manager Security and Secure Communications

You now have a choice of four security levels when using the job manager as your scheduler. These levels range from no security to user authentication requiring passwords to access jobs on the scheduler.

You also have a choice to use secure communications between the job manager and workers.

For more detailed descriptions of these features and information about setting up job manager security, see Set MJS Cluster Security.

The default setup uses no security, to match the behavior of past releases.

## Generic Scheduler Interface Enhancements

### Decode Functions Provided with Product

Generic scheduler interface decode functions for distributed and parallel jobs are now provided with the product. The two decode functions are named:

```
parallel.cluster.generic.distributedDecodeFcn
parallel.cluster.generic.parallelDecodeFcn
```

These functions are included on the workers' path. If your submit functions make use of the definitions in these decode functions, you do not have to provide your own decode functions. For example, to use the standard decode function for distributed jobs, in your submit function set `MDCE_DECODE_FUNCTION` to `'parallel.cluster.generic.distributedDecodeFcn'`. For information on using the generic scheduler interface with submit and decode functions, see Use the Generic Scheduler Interface.

### Enhanced Example Scripts

This release provides new sets of example scripts for using the generic scheduler interface. As in previous releases, the currently supported scripts are provided in the folder

*matlabroot*/toolbox/distcomp/examples/integration

In this location there is a folder for each type of scheduler:

- lsf — Platform LSF®
- pbs — PBS
- sge — Sun™ Grid Engine
- ssh — generic UNIX-based scripts
- winmpiexec — mpiexec on Windows

For the updated scheduler folders (lsf, pbs, sge), subfolders within each specify scripts for different cluster configurations: shared, nonshared, remoteSubmission.

For more information on the scripts and their updates, see the README file provided in each folder, or see Supplied Submit and Decode Functions.

## Compatibility Considerations

For those schedulers types with updated scripts in this release (lsf, pbs, sge), the old versions of the scripts are provided in the folder *matlabroot*/toolbox/distcomp/examples/integration/old. These old scripts might be removed in future releases.

## batch Now Able to Run Functions

Batch jobs can now run functions as well as scripts. For more information, see the batch reference page.

## batch and matlabpool Accept Scheduler Object

The batch function and the functional form of matlabpool now accept a scheduler object as their first input argument to specify which scheduler to use for allocation of compute resources. For more information, see the batch and matlabpool reference pages.

## Enhanced Functions for Distributed Arrays

### qr Supports Distributed Arrays

The qr function now supports distributed arrays. For restrictions on this functionality, type

```
help distributed/qr
```

### mldivide Enhancements

The mldivide function (\) now supports rectangular distributed arrays. Formerly, only square matrices were supported as distributed arrays.

When operating on a square distributed array, if the second input argument (or right-hand side of the operator) is replicated, `mldivide` now returns a distributed array.

## Compatibility Considerations

In previous releases, `mldivide` returned a replicated array when the second (or right-hand side) input was replicated. Now it returns a distributed array.

### chol Supports 'lower' option

The chol function now supports the `'lower'` option when operating on distributed arrays. For information on using `chol` with distributed arrays, type

```
help distributed/chol
```

### eig and svd Return Distributed Array

When returning only one output matrix, the eig and svd functions now return a distributed array when the input is distributed. This behavior is now consistent with outputs when requesting more than one matrix, which returned distributed arrays in previous releases.

## Compatibility Considerations

In previous releases, `eig` and `svd` returned a replicated array when you requested a single output. Now they return a distributed array if the output is a single matrix. The behavior when requesting more than one output is not changed.

### transpose and ctranspose Support 2dbc

In addition to their original support for 1-D distribution schemes, the functionsctranspose and transpose now support 2-D block-cyclic (`'2dbc'`) distributed arrays.

### Inf and NaN Support Multiple Formats

Distributed and codistributed arrays now support `nan`, `NaN`, `inf` and `Inf` for not-a-number and infinity values with the following functions:

| Infinity Value | Not-a-Number |
|---|---|
| codistributed.Inf | codistributed.NaN |
| `codistributed.inf` | `codistributed.nan` |
| distributed.Inf | distributed.NaN |
| `distributed.inf` | `distributed.nan` |

## Support for Microsoft Windows HPC Server 2008 R2

Parallel Computing Toolbox software now supports Microsoft Windows HPC Server 2008 R2. There is no change in interface compared to using HPC Server 2008. Configurations and other toolbox utilities use the same settings to support both HPC Server 2008 and HPC Server 2008 R2.

## User Permissions for MDCEUSER on Microsoft Windows

The user identified by the `MDCEUSER` parameter in the `mdce_def` file is now granted all necessary privileges on a Windows system when you install the mdce process. For information about what permissions are granted and how to reset them, see Set the User.

## Compatibility Considerations

In past releases, you were required to set the `MDCEUSER` permissions manually. Now this is done automatically when installing the mdce process.

# R2010a

## New Save and Load Abilities for Distributed Arrays

You now have the ability to save distributed arrays from the client to a single MAT-file. Subsequently, in the client you can load a distributed array from that file and have it automatically distributed to the MATLAB pool workers. The pool size and distribution scheme of the array do not have to be the same when you load the array as they were when you saved it.

You also can now load data directly into distributed arrays, even if the originally saved arrays were not distributed.

For more information, see the dsave and dload reference pages.

## Enhanced Functions for Distributed Arrays

The svd function now supports single-precision and complex data in distributed arrays. Other functions enhanced to support single-precision distributed arrays are chol, lu, mldivide, and eig.

In addition to their original support for 1-D distribution by columns, the enhanced tril and triu functions now support arrays with 1-D distribution by rows or when the distribution dimension is greater than 2, and 2-D block-cyclic (`'2dbc'`) distributed arrays.

## Importing Configurations Programmatically

A new function allows you to programmatically import parallel configurations. For more information, see the importParallelConfig reference page.

## Enhanced 2-D Block-Cyclic Array Distribution

2-D block-cyclic (`'2dbc'`) array distribution now supports column orientation of the lab grid. The codistributor2dbc function now accepts the value `'col'` for its orientation argument, which is reflected in the codistributor object's Orientation property. For information on `'2dbc'` distribution and using lab grids, see 2-Dimensional Distribution.

## New Remote Startup of mdce Process

New command-line functionality allows you to remotely start up MATLAB Distributed Computing Server processes on cluster machines from the desktop computer. For more information, see the remotemdce reference page.

## Obtaining mdce Process Version

An enhancement to the `mdce` command lets you get the command version by executing

```
mdce -version
```

For more information on this command, see the mdce reference page.

## Demo Updates

Product demos are available in the `Demos` node under `Parallel Computing Toolbox` in the help broswer.

### Benchmarking A\b

This new demo benchmarks Parallel Computing Toolbox performance with the `mldivide` function.

### BER Performance of Equalizer Types

Demos of BER Performance of Several Equalizer Types have been removed from Parallel Computing Toolbox, because Communications Toolbox now incorporates support for Parallel Computing Toolbox. See the Communications Toolbox release notes.

## taskFinish File for MATLAB Pool

The taskFinish file (`taskFinish.m`) for a MATLAB pool now executes when the pool closes.

## Compatibility Considerations

In previous versions of the software, the `taskFinish` file executed when a MATLAB pool opened. Beginning with this release, it runs when the pool closes.

# R2009b

## New Distributed Arrays

A new form of distributed arrays provides direct access from the client to data stored on the workers in a MATLAB pool. Distributed arrays have the same appearance and rules of indexing as regular arrays.

You can distribute an existing array from the client workspace with the command

```
D = distributed(X)
```

where `X` is an array in the client, and `D` is a distributed array with its data on the workers in the MATLAB pool. Distributing an array is performed outside an `spmd` statement, but a MATLAB pool must be open.

Codistributed arrays that you create on the workers within `spmd` statements are accessible on the client as distributed arrays.

The following new functions and methods support distributed arrays.

| Function Name | Description |
|---|---|
| distributed | Distribute existing array from client workspace to workers |
| distributed.rand, distributed.ones, etc. | Create distributed array consistent with indicated method, constructing on workers only |
| gather | Transfer data from MATLAB pool workers to client |
| isdistributed | True for distributed array |
| `C(x,y)` | Indexing into distributed array `C` on client to access data stored as codistributed arrays on workers |

## Renamed codistributor Functions

As part of the general enhancements for distributed arrays, several changes to the codistributed interface appear in this release.

## Compatibility Considerations

The following table summarizes the changes in function names relating to codistributed arrays.

| Old Function Name | New Function Name |
|---|---|
| `codcolon` | codistributed.colon |
| `codistributed(..., 'convert')` | codistributed`(...)` |
| `codistributed(...)` without `'convert'` option | codistributed.build |
| `codistributed(L, D)` using distribution scheme of `D` to define that of `L` | codistributed.build`(L, getCodistributor(D))` |
| codistributor`('1d', ...)` | Still available, but can also use codistributor1d |
| codistributor`('2d', ...)` | `codistributor('2dbc', ...)` or codistributor2dbc |
| `codistributor(arrayname)` | getCodistributor |
| `defaultLabGrid` | codistributor2dbc.defaultLabGrid |
| `defaultPartition` | codistributor1d.defaultPartition |
| `isa(X, 'codistributed')` | Still available, but can also use iscodistributed`(X)` |
| `localPart` | getLocalPart |
| `redistribute(D)` using default distribution scheme | redistribute`(D, codistributor())` |
| `redistribute(D1, D2)` using distribution scheme of `D2` to define that of `D1` | redistribute`(D1, getCodistributor(D2))` |

Some object methods are now properties:

| Old Method Name | New Property Name |
|---|---|
| `blockSize(codistObj)` | codistObj.BlockSize |
| `defaultBlockSize` | codistributor2dbc.defaultBlockSize |
| `distributionDimension(codistObj)` | codistObj.Dimension |
| `distributionPartition(codistObj)` | codistObj.Partition |
| `labGrid(codistObj)` | codistObj.LabGrid |

## Enhancements to Admin Center

Admin Center has several small enhancements, including more conveniently located menu choices, modified dialog boxes, properties dialog boxes for listed items, etc.

## Adding or Updating File Dependencies in an Open MATLAB Pool

Enhancements to the matlabpool command let you add or update file dependencies in a running MATLAB pool. The new forms of the command are

```
matlabpool('addfiledependencies', filedepCell)
matlabpool updatefiledependencies
```

where `filedepCell` is a cell array of strings, identical in form to those you use when adding file dependencies to a job or when you open a MATLAB pool. The `updatefiledependencies` option replicates any file dependency changes to all the labs in the pool.

## Updated globalIndices Function

The globalIndices function now requires that you specify the dimension of distribution as its second argument. Because this argument is required, it must precede the optional argument specifying the lab.

## Compatibility Considerations

In previous toolbox versions, the `globalIndices` function accepted the lab argument before the dimension argument, and both were optional. Now the dimension argument is required, and it must precede the optional lab argument.

## Support for Job Templates and Description Files with HPC Server 2008

Using job templates and job description files with Windows HPC Server 2008 lets you specify nodes and other scheduler properties for evaluating your jobs. To support these features, the ccsscheduler object has new properties:

- ClusterVersion — A string set to `'CCS'` or `'HPCServer2008'`
- JobTemplate — A string set to the name of the job template to use for all jobs

- JobDescriptionFile — A string set to the name of the XML file defining a base state for job creation

## Compatibility Considerations

CCS is now just one of multiple versions of HPC Server. While `'ccs'` is still acceptable as a type of scheduler for the findResource function, you can also use `'hpcserver'` for this purpose. In the Configurations Manager, the new scheduler type is available by selecting **File > New > hpcserver (ccs)**.

## HPC Challenge Benchmarks

Several new MATLAB files are available to demonstrate HPC Challenge benchmark performance. You can find the files in the folder *`matlabroot`*`/toolbox/distcomp/examples/benchmark/hpcchallenge`. Each file is self-documented with explanatory comments.

## pctconfig Enhanced to Support Range of Ports

The pctconfig function now lets you specify a range of ports for the Parallel Computing Toolbox client session to use. This range also includes ports used for a pmode session.

## Compatibility Considerations

You now specify the range of `pctconfig` ports with the `'portrange'` property; you no longer use the `'port'` property. As any client pmode session uses those ports of the `'portrange'` setting, you no longer use the `'pmodeport'` property.

## Random Number Generator on Client Versus Workers

The random number generator of the MATLAB workers now use a slightly different seed from previous releases, so that all the MATLAB workers and the client have separate random number streams.

## Compatibility Considerations

In past releases, while all the workers running a job had separate random number streams, the client had the same stream as one of the workers. Now the workers all have unique random number streams different from that of the client.

# R2009a

## Number of Local Workers Increased to Eight

You can now run up to eight local workers on your MATLAB client machine. If you do not specify the number of local workers in a command or configuration, the default number of local workers is determined by the value of the local scheduler's ClusterSize property, which by default is equal to the number of computational cores on the client machine.

## Compatibility Considerations

In previous versions, the default number of local workers was four, regardless of the number of cores. If you want to run more local workers than cores (for example, four workers with only one or two cores), you must set the value of `ClusterSize` equal to or greater than the number of workers you need. Then you can specify the increased number of workers in the appropriate command or configuration, or let your `ClusterSize` setting control the default number of workers.

## Admin Center Allows Controlling of Cluster Resources

When using the MathWorks job manager, the Admin Center GUI now allows you to start, stop, and otherwise control job managers and MATLAB workers on your cluster nodes. For more information about Admin Center, see Admin Center in the MATLAB Distributed Computing Server documentation.

## Compatibility Considerations

You can no longer start Admin Center from the MATLAB Desktop **Parallel** pull-down menu. You must start Admin Center from outside MATLAB by executing the following:

- *`matlabroot`*`/toolbox/distcomp/bin/admincenter` (on UNIX operating systems)
- *`matlabroot`*`\toolbox\distcomp\bin\admincenter.bat` (on Microsoft Windows operating systems)

## Support for Microsoft Windows HPC Server 2008 (CCS v2)

The parallel computing products now support Microsoft Windows HPC Server 2008 (CCS v2), including service-oriented architecture (SOA) job submissions. There is no change to the programming interface for `ccs` options, other than the addition of a new CCS scheduler object property, UseSOAJobSubmission. For implications to the installation of

the MATLAB Distributed Computing Server, see the online installation instructions at `http://www.mathworks.com/distconfig`.

## New Benchmarking Demos

New benchmarking demos for Parallel Computing Toolbox can help you understand and evaluate performance of the parallel computing products. You can access these demos in the Help Browser under the `Parallel Computing Toolbox` node: expand the nodes for `Demos` then `Benchmarks`.

## Pre-R2008b Distributed Array Syntax Now Generates Error

In R2008b, distributed array syntax was updated for codistributed arrays. In that release, the old form of the syntax still worked, but generated a warning. Now in R2009a, the old forms of the syntax no longer work and generate an error. For a summary of the syntax updates, see "Changed Function Names for Codistributed Arrays" on page 14-4.

## LSF Support on Mac OS X 10.5.x

For availability of Platform LSF support on Macintosh OS X 10.5.x, contact Platform Computing Corporation via their Web site at `http://www.platform.com/Products/platform-lsf/technical-information`. If Platform Computing does not support LSF on Mac OS X 10.5.x, then Parallel Computing Toolbox and MATLAB Distributed Computing Server cannot support this combination.

# R2008b

## MATLAB Compiler Product Support for Parallel Computing Toolbox Applications

This release offers the ability to convert Parallel Computing Toolbox applications, using MATLAB Compiler, into executables and shared libraries that can access MATLAB Distributed Computing Server. For information on this update to MATLAB Compiler, see Applications Created with Parallel Computing Toolbox Can Be Compiled.

### Limitations

- MATLAB Compiler does not support configurations that use the local scheduler or local workers (i.e., workers that run locally on the desktop machine running the MATLAB client session).
- Compiled Parallel Computing Toolbox applications do not support Simulink® software. For a list of other unsupported products, see the Web page http://www.mathworks.com/products/ineligible_programs/.
- When workers are running a task from compiled code, they can execute only compiled code and toolbox code. They cannot execute functions contained in the current directory. Batch and MATLAB pool jobs attempt to change the worker working directory to the client working directory. When noncompiled files in the current directory conflict with compiled versions (for example, files with different extensions), an error is thrown.

## spmd Construct

A new single program multiple data (spmd) language construct allows enhanced interleaving of serial and parallel programming, with interlab communication.

The general form of an `spmd` statement is:

```
spmd
    <statements>
end
```

The block of code represented by `<statements>` executes in parallel on workers in the MATLAB pool. Data on the labs is available for access from the client via Composite objects. For more information, see the spmd reference page and Distributing Arrays and Running SPMD.

## Compatibility Considerations

Because `spmd` is a new keyword, it will conflict with any user-defined functions or variables of the same name. If you have any code with functions or variables named `spmd`, you must rename them.

## Composite Objects

Composite objects provide direct access from the client (desktop) program to data that is stored on labs in the MATLAB pool. The data of variables assigned inside an spmd block is available via Composites in the client. When a MATLAB pool is open, you can also create Composites directly from the client using the Composite function. See also Distributing Arrays and Running SPMD.

## Configuration Validation

The Configurations Manager is enhanced with the capability for validating configurations. Open the Configurations Manager on the MATLAB Desktop by clicking **Parallel > Manage Configurations**. For more information, see Validate Profiles.

## Rerunning Failed Tasks

When using a job manager, if a task does not complete due to certain system failures, it can attempt to rerun up to a specified number of times. New properties of a task object to control reruns and access information about rerun attempts are:

- MaximumNumberOfRetries
- AttemptedNumberOfRetries
- FailedAttemptInformation

## Enhanced Job Control with Generic Scheduler Interface

The generic scheduler interface now allows you to cancel and destroy jobs and tasks and to investigate the state of a job. The following new properties of the generic scheduler object facilitate these features:

- `GetJobStateFcn`
- `DestroyJobFcn`

- `DestroyTaskFcn`
- `CancelJobFcn`
- `CancelTaskFcn`

New toolbox functions to accommodate this ability are:

- `getJobSchedulerData`
- `setJobSchedulerData`

For more information on this new functionality, see Manage Jobs with Generic Scheduler.

## Changed Function Names for Codistributed Arrays

What was known in previous releases as distributed arrays are henceforth called *codistributed* arrays. Some functions related to constructing and accessing codistributed arrays have changed names in this release.

## Compatibility Considerations

The following table summarizes the changes in function names relating to codistributed arrays. The first three functions behave exactly the same with no change in operation, arguments, etc. The `isa` function takes the argument `'codistributed'` in addition to the array in question.

| Old Function Name | New Function Name |
|---|---|
| `distributed` | `codistributed` |
| `distributor` | `codistributor` |
| `dcolon` | `codcolon` |
| `isdistributed` | `isa(X, 'codistributed')` |

## Determining if a MATLAB Pool is Open

The function `matlabpool` now allows you to discover if a pool of workers is already open. The form of the command is:

```
matlabpool size
```

For more information about this option and others, see the matlabpool reference page.

# R2008a

## Renamed Functions for Product Name Changes

As of result of the product name changes, some function names are changing in this release.

## Compatibility Considerations

Two function names are changed to correspond to the new product names:

- `dctconfig` has been renamed pctconfig.
- `dctRunOnAll` has been renamed pctRunOnAll.

## New batch Function

The new `batch` function allows you to offload work from the client to one or more workers. The batch submission can run scripts that can include jobs that distribute work to other workers. For more information, see the batch reference page, and Getting Started in the Parallel Computing Toolbox User's Guide.

### New Matlabpool Job

The batch functionality is implemented using the new MATLABpool job feature. A MATLAB pool job uses one worker to distribute a job to other workers, thereby freeing the client from the burden of tracking and job's progress and manipulating data. For more information, see the createMatlabPoolJob reference page.

## Enhanced Job Creation Functions

The createJob and createParallelJob functions have been enhanced to run without requiring a scheduler object as an argument. This is also true for the new createMatlabPoolJob function. When a scheduler is not specified, the function uses the scheduler identified in the applicable parallel configuration. For details, see the reference page for each function.

## Increased Data Size Transfers

The default size limitation on data transfers between clients and workers has been significantly increased. In previous releases the default limitation imposed by the JVM

memory allocation was approximately 50 MB. The new higher limits are approximately 600 MB on 32-bit systems and 2 GB on 64-bit systems. See Object Data Size Limitations.

## Changed Function Names for Distributed Arrays

Several functions related to distributed arrays have changed names in this release.

## Compatibility Considerations

The following table summarizes the changes in function names relating to distributed arrays.

| Old Function Name | New Function Name |
|---|---|
| `darray` | `distributed`, `distributor` |
| `distribute` | `distributed` |
| `dcolonpartition` | `defaultPartition` |
| `distribdim` | `distributionDimension` |
| `isdarray` | `isdistributed` |
| `labgrid` | `labGrid` |
| `local` | `localPart` |
| `partition` | `distributionPartition` |
| `localspan` | `globalIndices` |

## Support for PBS Pro and TORQUE Schedulers

Parallel Computing Toolbox software now fully supports PBS Pro® and TORQUE schedulers. These schedulers are integrated into parallel configurations and scheduler-related functions like findResource.

---

**Note** If you do not have a shared file system between client and cluster machines, or if you cannot submit jobs directly to the scheduler from the client machine, any use of third-party schedulers for parallel jobs (including pmode, matlabpool, and parfor) requires that you use the generic scheduler interface.

---

## findResource Now Sets Properties According to Configuration

The `findResource` function now sets the properties on the object it creates according to the configuration identified in the function call.

## Compatibility Considerations

In past releases, findResource could use a configuration to identify a scheduler, but did not apply the configuration settings to the scheduler object properties. If your code uses separate statements to find an object then set properties, this still works, but is not necessary any more.

## parfor Syntax Has Single Usage

The parfor statement is now recognized only for parallel `for`-loops, not for loops over a distributed range in parallel jobs.

## Compatibility Considerations

In R2007b, the pre-existing form of `parfor` was replaced by `for i = (drange)`, but both forms of syntax were recognized in that release. Now `parfor` has only one context, so `parfor` statements used in parallel jobs in code for versions prior to R2007a must be modified to use `for (drange)`.

#### Limitations

##### P-Code Scripts

You can call P-code script files from within a `parfor`-loop, but P-code script cannot contain a `parfor`-loop.

##### sim Inside parfor-Loops

Running simulations in a `parfor`-loop with the `sim` command at the top level of the loop is not allowed in this release. A `sim` command visible in a `parfor`-loop generates an error, although you can call `sim` inside a function that is called from the loop. Be sure that the various labs running simulations do not have the same working directory, as interference can occur with the simulation data.

## dfeval Now Destroys Its Job When Finished

When finished performing its distributed evaluation, the dfeval function now destroys the job it created.

## Compatibility Considerations

If you have any scripts that rely on a job and its data still existing after the completion of `dfeval`, or that destroy the job after `dfeval`, these scripts will no longer work.

# R2007b

Version: 3.2

New Features

Bug Fixes

Compatibility Considerations

## New Parallel for-Loops (parfor-Loops)

New parallel `for`-loop (`parfor`-loop) functionality automatically executes a loop body in parallel on dynamically allocated cluster resources, allowing interleaved serial and parallel code. For details of new parfor functionality, see Parallel for-Loops (parfor) in the Distributed Computing Toolbox™ documentation.

#### Limitations

#### P-Code Scripts

You can call P-code script files from within a `parfor`-loop, but P-code script cannot contain a `parfor`-loop.

## Compatibility Considerations

In past releases, `parfor` was a different function. The new parfor uses parentheses in defining its range to distinguish it from the old `parfor`.

New `parfor`:

```
parfor (ii = 1:N); <body of code>; end;
```

Old `parfor`:

```
parfor ii = 1:N; <body of code>; end;
```

For this release, the old form of `parfor` without parentheses is still supported, although it generates a warning. You can read more about the new form of this existing functionality in Using a for-Loop Over a Distributed Range (for-drange). You should update your existing `parfor` code to use the new form of `for`-loops over a distributed range (`for-drange`), thus,

```
for ii = drange(1:N); <body of code>; end;
```

## Configurations Manager and Dialogs

This release introduces a new graphical user interface for creating and modifying user configurations, and for designating the default configuration used by some toolbox functions. For details about the configurations manager, see Cluster Profiles in the Distributed Computing Toolbox documentation.

## Compatibility Considerations

This new feature has no impact on how configurations are used in a program, only on how configurations are created and shared among users. In previous versions of the product, you modified your configurations by editing the file *matlabroot*`/toolbox/distcomp/user/distcompUserConfig.m`. Now the configuration data is stored as part of your MATLAB software preferences.

The new configurations manager cannot directly import old-style configurations that were defined in the `distcompUserConfig.m` file. However, a utility called `importDistcompUserConfig`, available on the MATLAB Central Web site, allows you to convert and import your existing configurations into the new configurations manager. Visit `http://www.mathworks.com/matlabcentral` and search for `importDistcompUserConfig`.

## Default Configuration

This version of the toolbox enables you to select a user configuration to use as the default. Thus, commands such as pmode and matlabpool will use the default configuration without your having to specify it each time you run the command. You can set the default configuration using the configurations graphical interface, or programmatically with the defaultParallelConfig function.

## Parallel Profiler

A new parallel profiler graphical user interface generates reports on lab computation and communication times during execution of parallel jobs. For details about this new feature, see Profiling Parallel Code.

## MDCE Script for Red Hat Removed

The MDCE script `rh_mdce`, specific to Red Hat Linux®, has been removed from *matlabroot*`/toolbox/distcomp/util/bin`.

## Compatibility Considerations

If you make use of this script, you must replace it with its more generic equivalent,

*matlabroot*`/toolbox/distcomp/bin/mdce`.

# R2007a

Version: 3.1

New Features

Bug Fixes

Compatibility Considerations

## Local Scheduler and Workers

A local scheduler allows you to schedule jobs and run up to four workers or labs on a single MATLAB client machine without requiring engine licenses. These workers/labs can run distributed jobs or parallel jobs, including pmode sessions, for all products for which the MATLAB client is licensed. This local scheduler and its workers do not require a job manager or third-party scheduler.

## New pmode Interface

The interactive parallel mode (pmode) has a new interface. The pmode command input and displays of the lab outputs are provided in a user interface that you can separate from the MATLAB client Command Window.

## Compatibility Considerations

In previous versions of Distributed Computing Toolbox, the pmode interface used the MATLAB Command Window, with the pmode input using a different prompt. The output from the labs was intermingled with the MATLAB client output.

## New Default Scheduler for pmode

If you start pmode without specifying a configuration,

```
pmode start
```

pmode automatically starts a parallel job using the local scheduler with labs running on the client machine. For more information about running pmode, see Interactive Parallel Computation with pmode in the Distributed Computing Toolbox documentation.

## Compatibility Considerations

In the previous version of the toolbox, when pmode was started without specifying a configuration, it searched the network for the first available job manager to use as a scheduler.

## Vectorized Task Creation

The `createTask` function can now create a vector of tasks in a single call when you provide a cell array of cell arrays for input arguments. For full details, see the createTask reference page.

## Compatibility Considerations

In previous versions of the distributed computing products, if your task function had an input argument that was a cell array of cell arrays, your code will need to be modified to run the same way in this release.

For example, your old code may have been written as follows so that the function `myfun` gets four cell array input arguments:

```
createTask(j, @myfun, 1, {{C1} {C2} {C3} {C4}})
```

In this new version, the same code will produce four tasks. To get the old functionality, you must wrap the four cell arrays in another cell array, so that `createTask` knows to create only one task.

```
createTask(j, @myfun, 1, { {{C1} {C2} {C3} {C4}} })
```

## Additional Submit and Decode Scripts

There are several submit and decode functions provided with the toolbox for your use with the generic scheduler interface. These files are in the directory

*matlabroot*`/toolbox/distcomp/examples/integration`

This version of the toolbox includes new subdirectories for Platform LSF and PBS, to support network configurations in which the client and worker computers do not share a file system. For more information, see Supplied Submit and Decode Functions in the Distributed Computing Toolbox documentation.

## Jobs Property of Job Manager Sorts Jobs by ID

The Jobs property of a job manager object now contains the jobs in the order in which they were created, as indicated by the ID property of each job. Similarly, the findJob

function returns jobs sequenced by their `ID`, unless otherwise specified. This change makes job manager behavior consistent with the behavior of third-party schedulers.

## Compatibility Considerations

In previous versions of the distributed computing products, when using a job manager, jobs were arranged in the `Jobs` property or by `findJob` according to the status of the job.

## New Object Display Format

When you create distributed computing objects (scheduler, job, or task) without a semicolon at the end of the command, the object information is displayed in a new format. This new format is also shown when you use the display function to view an object or simply type the object name at the command line.

## Compatibility Considerations

With this enhancement, the output format shown when creating an object has changed.

## Enhanced MATLAB Functions

Several MATLAB functions have been enhanced to work on distributed arrays:

- cat
- find
- horzcat
- subsindex
- vertcat

For a complete list of MATLAB functions that are enhanced to work on distributed arrays, see Using MATLAB Functions on Codistributed Arrays in the Distributed Computing Toolbox documentation.

## darray Function Replaces distributor Function

The function `darray` now defines how an array is distributed among the labs in a parallel job.

## Compatibility Considerations

In the previous version of the toolbox, the `distributor` function was used to define how an array was distributed. In many cases, you can replace a call to `distributor` with a call to `darray`. For example, if you used `distributor` without arguments as an input to an array constructor,

```
rand(m, n, distributor());
```

you can update the code to read,

```
rand(m, n, darray());
```

## rand Seeding Unique for Each Task or Lab

The random generator seed is now initialized based on the task ID for distributed jobs, or the `labindex` for parallel jobs (including pmode). This ensures that the set of random numbers generated for each task or lab within a job is unique, even when you have more than 82 tasks or labs.

## Compatibility Considerations

In the previous version of the distributed computing products, the `rand` function could by default generate the same set of numbers for some tasks or labs when these exceeded 82 for a job.

## Single-Threaded Computations on Workers

Despite the ability in MATLAB software to perform multithreaded computations on multiple-CPU machines, the workers and labs running distributed and parallel jobs perform only single-threaded computations, so that multiprocessor cluster machines can better accommodate multiple workers or labs.

# R2006b

## Support for Windows Compute Cluster Server (CCS)

Distributed Computing Toolbox software and MATLAB Distributed Computing Engine™ software now let you program jobs and run them on a Windows Compute Cluster Server. For information about programming in the toolbox to use Windows Compute Cluster Server (CCS) as your scheduler, see the findResource reference page.

## Windows 64 Support

The distributed computing products now support Windows 64 (Win64) for both MATLAB client and MATLAB worker machines.

## Parallel Job Enhancements

### Parallel Jobs Support Any Scheduler

Support for parallel jobs now extends to any type of scheduler. In previous releases, only the MathWorks® job manager and mpiexec scheduler object supported parallel jobs. You can now run parallel jobs on clusters scheduled by a job manager, Windows Compute Cluster Server (CCS), Platform LSF, mpiexec, or using the generic scheduler interface. For programming information, see Program Communicating Jobs.

### New labSendReceive Function

The `labSendReceive` function is introduced in this release. This function performs the same things as both `labSend` and `labReceive`, but greatly reduces the risk of deadlock, because the send and receive happen simultaneously rather than by separate statements. For more information, see the labSendReceive reference page.

### Improved Error Detection

This release offers improved error detection for miscommunication between labs running parallel jobs. Most notable among the improvements are error detection of mismatched `labSend` and `labReceive` statements.

## Distributed Arrays

Distributed arrays are partitioned into segments, with each segment residing in the workspace of a different lab, so that each lab has its own array segment to work with. Reducing the size of the array that each lab has to store and process means a more

efficient use of memory and faster processing, especially for large data sets. For more information, see “Distributed Arrays and SPMD”.

#### parfor: Parallel for-Loops

Parallel for-loops let you run a for-loop across your labs simultaneously. For more information, see Using a for-Loop Over a Distributed Range (for-drange) or the parfor reference page.

## Interactive Parallel Mode (pmode)

The interactive parallel mode (pmode) lets you work interactively with a parallel job running simultaneously on a number of labs. Commands you type at the pmode command line are executed on all labs at the same time. Each lab executes the commands in its own workspace on its own local variables or segments of distributed arrays. For more information, see Run Parallel Jobs Interactively Using pmode.

## Moved MDCE Control Scripts

To provide greater consistency across all platforms, the MDCE control scripts for Windows have moved and those for UNIX and Macintosh have new names.

## Compatibility Considerations

#### Windows Utilities Moved

In previous versions of the distributed computing products, the MDCE utilities for Windows computers were located in

```
matlabroot\toolbox\distcomp\bin\win32
```

The utilities are now located in

```
matlabroot\toolbox\distcomp\bin
```

The files that have moved are

```
nodestatus
mdce
startjobmanager
stopjobmanager
startworker
```

```
stopworker
mdce_def.bat
```

**UNIX and Macintosh Utilities Renamed**

In previous versions of the distributed computing products, the MDCE utilities for UNIX and Macintosh computers were called by

```
nodestatus.sh
startjobmanager.sh
stopjobmanager.sh
startworker.sh
stopworker.sh
```

You can now call these with the following commands:

```
nodestatus
startjobmanager
stopjobmanager
startworker
stopworker
```

---

**Note:** For UNIX and Macintosh, `mdce` and `mdce_def.sh` have not been moved or renamed.

---

## rand Seeding Unique for Each Task or Lab

The random generator seed is now initialized based on the task ID for distributed jobs, or the `labindex` for parallel jobs (including pmode). This ensures that the random numbers generated for each task or lab are unique within a job.

## Compatibility Considerations

In previous versions of the distributed computing products, the `rand` function would by default generate the same set of numbers on each worker.

## Task ID Property Now Same as labindex

Although you create only one task for a parallel job, the system copies this task for each worker that runs the job. For example, if a parallel job runs on four workers (labs), the

Tasks property of the job contains four task objects. The first task in the job's `Tasks` property corresponds to the task run by the lab whose labindex is `1`, and so on, so that the ID property for the task object and `labindex` for the lab that ran that task have the same value. Therefore, the sequence of results returned by the getAllOutputArguments function corresponds to the value of `labindex` and to the order of tasks in the job's `Tasks` property.

## Compatibility Considerations

In past releases, there was no correlation between `labindex` and the task `ID` property.