

MathLink for AppleScript

Tutorial and Command Reference

Version 1.3

© 2004–2005 uni software plus

License Agreement

This License Agreement is a legal agreement between the user and uni software plus GmbH, which owns a proprietary computer software and "on-line" or electronic documentation (collectively known as "MathLink for AppleScript"). By installing, copying, or otherwise using the Software, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, do not install, copy, or use the Software.

1. Copyright Notice

THIS SOFTWARE IS COPYRIGHT © 2004–2005 BY uni software plus GmbH.
PORTIONS © 2002 Wolfram Research, Inc.

2. Grant of Use License

YOU MAY USE AND/OR DISTRIBUTE THE SOFTWARE WITHOUT RESTRICTION INCLUDING COMMERCIAL APPLICATIONS. THE ORIGIN OF THIS SOFTWARE MUST NOT BE MISREPRESENTED; YOU MUST NOT CLAIM THAT YOU WROTE THE ORIGINAL SOFTWARE. IF YOU BUNDLE THE SOFTWARE WITH ANY COMMERCIAL OFFERING, YOU MUST DISPLAY THE COPYRIGHT NOTICE IN THE SOFTWARE'S DOCUMENTATION AND ABOUT BOX. THIS NOTICE MAY NOT BE REMOVED OR ALTERED FROM ANY DISTRIBUTION.

3. Disclaimer of Warranty

THIS SOFTWARE IS PROVIDED AS-IS, WITHOUT WARRANTY OF ANY KIND. uni software plus GmbH DISCLAIMS ANY DAMAGES RESULTING FROM THE USE OR MISUSE OF THIS SOFTWARE.

In case of any questions concerning this product please contact us at:

uni software plus GmbH

Kreuzstrasse 15a

A-4040 Linz / Austria

Email: mathlinkosax@unisoftwareplus.com

WWW: <http://www.unisoftwareplus.com/products/mathlinkosax/>

© 2004–2005 uni software plus. All Rights reserved.

Mathematica, MathLink and gridMathematica are registered trademarks of Wolfram Research, Inc.

AppleScript is a registered trademark of Apple Computer, Inc.

Contents

1	Introduction	5
1.1	What is MathLink for AppleScript?	5
1.2	System Requirements	5
1.3	Installation	5
1.4	Requirements	6
1.5	What's New in this Release?	6
	New and Changed Sections in Release 1.3 of the Documentation	6
	New and Changed Sections in Release 1.2 of the Documentation	6
	New and Changed Sections in Release 1.1 of the Documentation	7
2	Using MathLink For AppleScript	8
2.1	A Simple Example	8
2.2	Mathematica Expressions in AppleScript	9
2.3	Expressions and Packets	12
2.4	Calling AppleScript from the Mathematica Kernel	14
	A Simple Example	14
	Including Accessory Mathematica Code	15
2.5	Using AppleScript applets as MathLink-Compatible Programs	17
	Requesting Evaluations by the Kernel	19
	Error Handling	19
2.6	Using a Remote Mathematica Kernel	21
	Launch and Connect via Remote Apple Events	21
	Launch via Remote Apple Events and Connect via MathLink	23
	Launch via Secure Shell and Connect via MathLink	24
3	Command Reference	26
3.1	Overview	26
3.2	MathLink create	26
3.3	MathLink connect	28
3.4	MathLink close	29
3.5	MathLink activate	29
3.6	MathLink launch	30
3.7	MathLink links	30
3.8	MathLink read	31
3.9	MathLink write	32
3.10	MathLink peek	32
3.11	MathLink ready	33
3.12	MathLink evaluate	33

3.13 MathLink enter	35
3.14 MathLink abort	36
3.15 MathLink interrupt	37
3.16 MathLink terminate	38
3.17 MathLink install	38
3.18 MathLink handle	40

1 Introduction

1.1 What is MathLink for AppleScript?

MathLink for AppleScript is a Scripting Addition (OSAX) that enables you to use Mathematica as a computational engine in AppleScript programs. MathLink for AppleScript is based on *MathLink*, a communication protocol for Mathematica, i.e., a way of sending data and commands back and forth between Mathematica and other programs.

By combining the powers of Mathematica and AppleScript, you can manipulate and visualize data with over a thousand of Mathematica's built-in functions and options. It is easy to send even large data sets from applications to Mathematica via AppleScript for sophisticated analysis and then to use the results right in your AppleScript enabled application. Because Mathematica itself is extensible, you can add even more functionality through Mathematica Applications Library add-ons such as Control Systems Professional and Optica.

The second use of MathLink for AppleScript is to allow external functions written in AppleScript to be called from within the Mathematica environment. This makes it easy to extend Mathematica with functionality provided by AppleScript enabled applications.

1.2 System Requirements

To use MathLink for AppleScript you need:

- A Macintosh with a PowerPC processor.
- Mac OS X 10.2 (Jaguar) or later.
- Mathematica 4.2 or later.

1.3 Installation

To install MathLink for AppleScript:

- Quit all running applications
- Copy the file MathLink Scripting.osax
in the local domain of your machine, into:

/Library/ScriptingAdditions/

or, in your user domain, into:

~/Library/ScriptingAdditions/

If a `ScriptingAdditions` folder does not exist at the installation location, create one with that exact name.

Installing the scripting addition into the local domain requires that you have administrator rights on your machine.

If a previous version of `MathLink Scripting.osax` is already installed, the system may not let you replace it with the new file. In that case move the old file to the trash, then copy the new one.

1.4 Requirements

This document is not a tutorial on AppleScript but assumes that you are familiar with AppleScript. If you are new to AppleScript, here are two publications we want to recommend:

- *AppleScript: The Definite Guide* (O'Reilly & Associates)
- *AppleScript Language Guide* (Apple Technical Publications)

This document is not a tutorial on Mathematica, either. We assume that you are familiar with using Mathematica. If you are new to Mathematica, we recommend walking thru the ten-minute tutorial that is part of Mathematica's online help.

You will find the contents of the document easier to understand, if you have an idea about the communication protocol MathLink. We therefore recommend reading Todd Gayley's excellent MathLink tutorial. You can download this document from the Mathematica Information Center at Wolfram Research:

<http://library.wolfram.com/infocenter/TechNotes/174/>

1.5 What's New in this Release?

New and Changed Sections in Release 1.3 of the Documentation

- Added description of new command [MathLink peek](#) on page 32.
- Updated AppleScript code sections to use Mathematica version 5.2.

New and Changed Sections in Release 1.2 of the Documentation

- Added section "Using a Remote Mathematica Kernel" on page 21.
- Added description of new command [MathLink activate](#) on page 29.
- Updated AppleScript code sections to use Mathematica version 5.1.

New and Changed Sections in Release 1.1 of the Documentation

- Added description of new commands [MathLink abort](#) and [MathLink terminate](#) on page 36.
- Revised documentation of command [MathLink interrupt](#) on page 37 for additional parameter [waiting for response](#).
- Added new parameter [with host](#) to description of command [MathLink create](#) on page 26.

2 Using MathLink For AppleScript

2.1 A Simple Example

Let's look at a simple AppleScript that demonstrates how you can use MathLink for AppleScript to calculate the prime factors of the integer 127341272 with Mathematica:

```
set link to MathLink launch
    "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
    -mathlink"
MathLink evaluate "FactorInteger[127341272]" on link
MathLink close link
```

When this AppleScript is executed in Script Editor, it produces the following output in the event log pane:

```
MathLink launch
    "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
    -mathlink"
->{
    class:expression,
    head: |LinkObject|,
    elements:{
        "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
        -mathlink", 4
    }
}
MathLink evaluate "FactorInteger[127341272]" on link
->{{2, 3}, {37, 1}, {67, 1}, {6421, 1}}
MathLink close link
```

The first statement uses the command [MathLink launch](#) to launch the Mathematica kernel as a sub process and establishes a MathLink connection to it. The Mathematica kernel is the shell executable, which actually performs computations. When you launch Mathematica by double clicking its icon in the Finder, you are actually looking at the “front end” to Mathematica, which lets you edit Mathematica notebook documents. The front end uses the protocol MathLink to send expressions to the Mathematica kernel, too.

In order for the [MathLink launch](#) command to succeed, the full path to the Mathematica kernel must be specified as a valid POSIX path (i.e., using "/" and not ":" as a path delimiter). The example above assumes that Mathematica version 5.2 is installed in the Applications folder of the local domain. If your Mathematica application resides at a different location or your version of Mathematica is different than 5.2, please adjust the POSIX path accordingly.

The Mathematica kernel executable resides in Mathematica's application package in the folder `Mathematica 5.2.app/Contents/MacOS/`. You can inspect the contents of the Mathematica application package by selecting its icon and the Finder and choosing the command `Show Package Contents` from the Contextual Menu.

The Mathematica kernel is passed the argument `-mathlink` upon launch. This puts the kernel in the so-called *MathLink mode*, which is the usual mode for running Mathematica as a subprogram from an external program or front end.

The [MathLink launch](#) command returns an AppleScript object of type [expression](#) (see section 2.2 below) that represents the active MathLink connection to the kernel.

The second statement [MathLink evaluate](#) sends a textual expression to the Mathematica kernel for evaluation via the established MathLink connection. The statement uses Mathematica's built-in function `FactorInteger` to compute the prime factors of the integer 127341272. The statement returns a list of pairs, where the first item is the prime factor and the second item is the factor's exponent. MathLink for AppleScript takes care of parsing the output expression returned by the kernel and transforms it into an AppleScript list of integers, which can be easily processed.

The final statement [MathLink close](#) closes the MathLink connection and quits the Mathematica kernel process.

2.2 Mathematica Expressions in AppleScript

The smallest unit of information that can be exchanged using the AppleScript commands from MathLink for AppleScript is an *expression*. Expressions are the main type of data in Mathematica.

In Mathematica an expression can be written in the form `h[e1, e2, ...]`, where `h` is known as the *head* of the expression and the `ei` are known as the *elements* of the expression. Both the head and the elements may be expressions themselves.

All expressions are ultimately made up from a small number of basic or atomic types of objects:

Mathematica Type	Description
Integer	An integer number containing any number of digits.
Real	A real number of arbitrary precision
Symbol	Basic named object in Mathematica. Sequence of letters, letter-like forms and digits, not starting with a digit.
String	Character string containing any sequence of Unicode characters.

As an example the expression `1 + 2 * a * b` correspond to the Mathematica expression `Plus[1, Times[2, a, b]]`. An example of an

expression, where the head itself is an expression, is $(a + b)[\text{"Mathematica"}]$ which can be re-written as `Plus[a, b][\text{"Mathematica"}]`.

MathLink for AppleScript tries to map Mathematica expressions to AppleScript expressions with equal semantics. Due to the more general nature of Mathematica expressions, a suitable AppleScript type is not always available. In this case the new AppleScript record data type [expression](#) is used instead. The following table shows the type mappings supported:

Mathematica Type/Expression	AppleScript Type	Constraints
Integer	integer	Value is between $\pm(2^{29} - 3)$, the largest value expressible as an integer in AppleScript.
Real	real	Value must be expressible as an IEEE 64Bit floating point number.
Symbol	reference to AppleScript variable	Symbol names with letter-like forms will be shown between vertical bars (" ") in AppleScript.
String	unicode text	None.
List[e_1, e_2, \dots]	list	None.
True	true	None.
False	false	None.
Null	missing value	None.
Point[List[x, y]]	point	x and y must be integer values between $\pm(2^{15} - 1)$.
Rectangle[List[x_{min}, y_{min}], List[x_{max}, y_{max}]]	bounding rectangle	$x_{min}, y_{min}, x_{max}$ and y_{max} must be integer values between $\pm(2^{15} - 1)$.
List[Rule[lhs_1, rhs_1], Rule[lhs_2, rhs_2], ...]	record	Each lhs_i must be an expression of type Symbol or String.

The AppleScript record type [expression](#), which is defined in dictionary of the scripting addition, mimics the general structure of a Mathematica expression. This type can represent any Mathematica expression:

Class expression: a Mathematica expression.

Plural form:

[expressions](#)

Properties:

[head](#) anything -- The head of the expression.
[elements](#) a list of anything -- The elements of the expression.

Examples of expressions

The following AppleScript example shows that for small integer values the result of the evaluation of the Mathematica function `Factorial` is returned as an AppleScript [integer](#) value:

MathLink evaluate "Factorial[12]" on link
->479001600

If the result of the `Factorial` is too large to be representable with AppleScript's [integer](#) type, the common type [expression](#) is returned instead:

[MathLink evaluate](#) "Factorial[123]" [on](#) [link](#)

```
-> {  
    class: expression,  
    head: integer,  
    elements: {  
        "121463043670253296757662432418812958554542170884833823153  
289181618292358923621676688311569606126402021707358352212940  
477825910915704116514721860295199062616467307339074198149529  
600000000000000000000000000000"  
    }  
}
```

The actual integer value is contained in the `elements` property of the `expression` record as a string, which is capable of holding any number of digits.

By specifying the additional parameter [as expression](#) when calling [MathLink evaluate](#) you can prevent the result from being cast to a suitable AppleScript type. Regardless of the actual Mathematica result type, the AppleScript result will always be returned as an [expression](#) then:

MathLink evaluate "N[Pi, 50]" on link as expression

```
-> {
  class: expression,
  head: real,
  elements: {
    "3.1415926535897932384626433832795028841971693993751"
  }
}
```

Hint: the Mathematica function `N[expr, n]` gives the result with n-digit precision.

You can also use the [expression](#) type to directly build an expression to be evaluated by the Mathematica kernel. The following example shows how to use Mathematica to reverse an AppleScript list:

```
set theList to {1, 2, "3", 4.3}
MathLink evaluate {class:expression, head:a reference to |Reverse|,
elements:{theList}} on link
->{4.3, "3", 2, 1}
```

The example uses Mathematica's built-in function `Reverse`. Please note that the AppleScript clause **a reference to** is necessary here to prevent the AppleScript interpreter from trying to resolve the variable `|Reverse|`, which is only used as a means to represent the Mathematica symbol `Reverse`.

Some Mathematica functions return a list of rules as result (e.g., `{x -> 2, y -> "foo"}`). The scripting addition converts this list of rules to an AppleScript record, if the left hand side of each rule consists of an expression of type `Symbol` as in the following example:

[MathLink evaluate](#) "Solve[{x + 2*y == 4, 2*x - y == 8}, {x, y}]" [on link](#) -> `{{x:4, y:0}}`

The Mathematica function `Solve` in the example attempts to solve the set of equations for the variables `x` and `y` and returns a list of solutions. Each solution is given in terms of lists of rules. In AppleScript you can access the solution for each variable as follows:

```
repeat with solution in result
    ... x of solution ...
    ... y of solution ...
end repeat
```

2.3 Expressions and Packets

All data sent via MathLink are in the form of Mathematica expressions. Any Mathematica expression can be sent through MathLink.

Expressions sent to the Mathematica kernel and received from the kernel may convey different kinds of information. For example, an expression received from the kernel may be the result of a calculation, an error message or graphics display data. To distinguish different kinds of information, Mathematica uses the concept of *packets*.

A packet is an expression that contains a particular kind of information, wrapped with a symbol head that identifies what type of information is enclosed. Mathematica uses packets automatically when you run it in MathLink mode, which is the usual mode for running the Mathematica kernel as a subprogram from an external program. An external program that receives packets from Mathematica can read the packet type first and then dispatch to code written to handle that type of packet.

The scripting addition's dictionary defines an enumeration type for the different packet heads supported by the MathLink protocol. The following table lists the most important packet types:

Packet (Symbol Head)	AppleScript enumeration constant	Description
<code>EvaluatePacket</code>	evaluatepkt	An expression sent purely for

Packet (Symbol Head)	AppleScript enumeration constant	Description
		evaluation.
ReturnPacket	returnpkt	An expression returned from an evaluation.
EnterExpressionPacket	enterexprpkt	An expression to enter corresponding to an input line.
ReturnExpressionPacket	returnexprpkt	An expression returned corresponding to an output line.
EnterTextPacket	entertextpkt	Text to enter corresponding to an input line.
ReturnTextPacket	returntextpkt	Text returned corresponding to an output line.
MessagePacket	messagepkt	Mathematica message identifier (symbol::string).
TextPacket	textpkt	Text output from Mathematica, as produced by Print etc.
SyntaxPacket	syntaxpkt	Position at which a syntax error was detected in the input line.
DisplayPacket	displaypkt	Parts of a PostScript graphics.
DisplayEndPacket	displayendpkt	End of a PostScript graphics.
InputNamePacket	inputnamepkt	Text returned for the name of an input line (e.g., In[1]:=).
OutputNamePacket	outputnamepkt	Text returned for the name of the output line (e.g., Out[1]=).

You can use the appropriate enumeration constant to tag expressions as packets when sending data to the Mathematica kernel with the command [MathLink write](#) as in the following example:

```
MathLink write ->
  {class:expression, ->
    head:evaluatepkt, ->
    elements:{ ->
      {head: a reference to |Sqrt|, elements:{133.12314}}} ->
    } ->
  } ->
  to link
```

A subsequent call of [MathLink read](#) reads the result of the request to evaluate `Sqrt[133.12314]` as a [returnpkt](#) from the link.

```
MathLink read link
->{
  class:expression,
  head:returnpkt,
  elements:{
    11.537900155574
  }
}
```

2.4 Calling AppleScript from the Mathematica Kernel

Another use of MathLink for AppleScript is to allow you to call functions written in AppleScript from within Mathematica. This makes it easy to extend Mathematica with functionality provided by AppleScript enabled applications.

A Simple Example

Calling AppleScript from Mathematica is based on Mathematica's `Install` mechanism for external programs. Let's look at a trivial example of an AppleScript handler that we will modify in several ways to make it installable as a Mathematica function:

```
on addtwo(a, b)
    return a + b
end addtwo
```

The first step is to wrap the handler into a script object and to setup the definition of the function in Mathematica by adding the two properties `addtwo_Pattern` and `addtwo_Arguments`:

```
script MyHandler
    property addtwo_Pattern : "AddTwo[i_Integer, j_Integer]"
    property addtwo_Arguments : "{ i, j }"
    on addtwo(a, b)
        return a + b
    end addtwo
end script
```

The property `addtwo_Pattern` pattern will become the left-hand side of a function definition, exactly as you would type it if you were creating the entire function in Mathematica.

The `addtwo_Arguments` property specifies the expressions to be passed to the handler `addtwo`. These expressions don't have to be the same as the variable names used in the `addtwo_Pattern` property, although they often will be. You could, for example, use `{i * 2, Abs[j]}` instead. The number of argument expressions must correspond to the number of positional parameters declared in the AppleScript handler.

The point is that what you declare in the `addtwo_Pattern` line and the `addtwo_Arguments` property is Mathematica code; it will be used verbatim in a definition that could be sketched as follows:

```
AddTwo[i_Integer, j_Integer] :=
    SendToExternalProgramAndWaitForAnswer[{i, j}]
```

The second step is to install `AddTwo` in Mathematica with the `Install` function:

```
set link to MathLink launch
"/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
```

```

-mathlink -batchoutput"
MathLink write {class:expression, head:entertextpkt,
               elements:{"Install[$ParentLink]"}} to link
MathLink install MyHandler on link
MathLink read link -- reads result from Install[$ParentLink] call above

```

Using [MathLink launch](#) the Mathematica kernel is launched as a sub process in MathLink mode. The additional command line option [batchoutput](#) prevents the Mathematica kernel from generating unnecessary [inputnamepkt](#) and [outputnamepkt](#) packets.

The [MathLink write](#) command sends an [entertextpkt](#) object to the Mathematica kernel, which makes the kernel evaluate the Mathematica function `Install[$ParentLink]`. This function prepares the kernel for setting up definitions of external functions on our link. `$ParentLink` represents the connection between the Mathematica kernel and the AppleScript on the Mathematica kernel side of the link.

The actual definition of the function `AddTwo` is then conveyed by means of the [MathLink install](#) command. [MathLink install](#) uses AppleScript's reflection API to look for suitable handlers defined in the script object passed as direct parameter. A script handler is considered suitable for installation, if both a [_Pattern](#) and a [_Arguments](#) property prefixed by the handler name have been declared in the script object. [MathLink install](#) then sends the necessary definitions to the kernel, which completes the installation of the external function. The command returns the list of handler names that have been installed.

The external function `AddTwo` can now be called as if it were built into the Mathematica kernel:

```
MathLink evaluate "AddTwo[10, 20]" on link
```

The `AddTwo` function call will actually make the Mathematica kernel send back a [callpkt](#) object over the link, which will be transparently handled by the scripting addition by translating it into a call of the AppleScript handler [addtwo](#). The scripting addition also takes care of marshaling the function arguments and the function result.

Including Accessory Mathematica Code

Besides the [_Pattern](#) and [_Arguments](#) properties you can also specify arbitrary Mathematica code to be sent along with the definition of the external function. You might have some accessory code that your functions need to have exist in Mathematica.

You can specify arbitrary Mathematica code to be sent to the kernel before the external function is defined by adding a [_Begin](#) property prefixed by the handler name. A simple example is a usage message:

```

script MyHandler
  property addtwo_Pattern : "AddTwo[i_Integer, j_Integer]"

```

```

property addtwo_Arguments : "{ i, j }"
property addtwo_Begin : "AddTwo::usage = \"AddTwo[x, y] gives the
                        sum of two machine integers x and y.\""
on addtwo(a, b)
    return a + b
end addtwo
end script

```

A property `_End` prefixed by the handler name will be sent to the kernel after the corresponding handler has been installed:

```

script MyHandler
property addtwo_Pattern : "AddTwo[i_Integer, j_Integer]"
property addtwo_Arguments : "{ i, j }"
property addtwo_Begin : "AddTwo::usage = \"AddTwo[x, y] gives the
                        sum of two machine integers x and y.\""
on addtwo(a, b)
    return a + b
end addtwo
property addtwo_End : "AddTwo[{i_Integer, j_Integer}]:=AddTwo[i, j]"
end script

```

The above example uses Mathematica code in the property `addtwo_End` to make the function `AddTwo` work with a list of two integer values, too (e.g., `AddTwo[List[10, 20]]`).

Another common use is to make your handlers appear in a package context. The Mathematica function `Install` causes all functions defined in installable programs to appear in the `Global`` context. If you want the `AddTwo` function to appear in a package context instead, add an `Evaluate_Begin` and an `Evaluate_End` property to the script object definition:

```

script MyHandler
property Evaluate_Begin : "BeginPackage[\"MyPackage`\"]"
property addtwo_Pattern : "AddTwo[i_Integer, j_Integer]"
property addtwo_Arguments : "{ i, j }"
property addtwo_Begin : "AddTwo::usage = \"AddTwo[x, y] gives the
                        sum of two machine integers x and y.\""
on addtwo(a, b)
    return a + b
end addtwo
property addtwo_End : "AddTwo[{i_Integer, j_Integer}]:=AddTwo[i, j]"
property Evaluate_End : "EndPackage[ ]"
end script

```

Recapitulating the following listing shows the AppleScript event log generated by the example:

```

set link to MathLink launch
"/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel'
-mathlink -batchoutput"
->{class:expression, head:|LinkObject|,
  elements:{

```



```

"/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel'
-mathlink -batchoutput",
12}}
MathLink install MyHandler on link
-> {"addtwo"}
MathLink read link
-> {class: expression, head: returntextpkt,
    elements: {"LinkObject["ParentLink", 1, 1]}}
MathLink evaluate "Context[AddTwo]" on link
-> "MyPackage`"
MathLink evaluate "AddTwo[10, 20]" on link
-> 30
MathLink evaluate "AddTwo[{10, 20}]" on link
-> 30
MathLink close link

```

2.5 Using AppleScript applets as MathLink-Compatible Programs

Using the machinery provided by MathLink for AppleScript, you can create AppleScript applets that can be run as MathLink-compatible programs, i.e., programs installed with the `Install["prog"]` command in a Mathematica front-end session.

To create an AppleScript applet, launch Script Editor, create a new document and enter the following code (the `MyHandler` code is the same as in the previous section and has been omitted):

```

global gLink

script MyHandler
...
end script

on run
    set gLink to MathLink connect
    MathLink install MyHandler on gLink
end run

on quit
    MathLink close gLink
    continue quit
end quit

on idle
    try
        if MathLink ready gLink then
            MathLink handle gLink
        end if
    on error
        quit
    end try

```

```
return 0.2 -- call idle handler again in 0.2 seconds
end idle
```

Save the document with the name "Sample.app". Choose the file format Application and check the check box Stay Open in the Save panel.

The applet's [run](#) handler will be called, when the program is launched from the Mathematica front end by invoking the `Install` command. The [MathLink connect](#) command establishes a MathLink connection to the parent process, which happens to be the Mathematica kernel process. The [MathLink install](#) command then sets up the AppleScript handlers of the script object `MyHandler` as external Mathematica functions.

The [quit](#) handler will be called when the operating system quits the AppleScript applet. The quit handler code ensures that the MathLink connection is closed before continuing with the quit.

The applet needs to monitor its MathLink connection for incoming [callpkt](#) object from the Mathematica kernel. This is done in the [idle](#) handler of the applet. AppleScript sends the script applet periodic idle commands whenever it's not responding to incoming events.

The [idle](#) handler checks if there is input pending on the link using [MathLink ready](#) and handles a [callpkt](#) object sent by the kernel with the command [MathLink handle](#). The call will raise an error if the Mathematica kernel has closed the link (e.g., because `Uninstall` has been called on the Mathematica side). The error handler simply quits the applet.

To test the applet copy it to the folder `~/Library/Mathematica/Applications` (the library folder in your home folder). Then launch the Mathematica front end, create a new notebook and enter the following code (the generated Mathematica output is shown below the input line):

```
link = Install["Sample.app"]
-> LinkObject[/Users/kratky/Library/Mathematica/
  Applications/Sample.app, 2, 2]
```

The `Install` command launches the AppleScript applet and connects to it via MathLink. The call returns a `LinkObject` representing the MathLink connection it is using.

```
LinkPatterns[link]
-> {AddTwo[i_Integer, j_Integer]}
```

`LinkPatterns[link]` gives a list of the patterns defined when the link was setup.

```
?AddTwo
-> AddTwo[x, y] gives the sum of two machine
  integers x and y.
```

The input `?AddTwo` prints the usage message for the command.

```
AddTwo[21, 21]
-> 42
```

Evaluating the `AddTwo` function now makes the Mathematica call the AppleScript applet via MathLink. Please note that the fact that `Sample.app` is written in AppleScript is completely transparent to the Mathematica kernel.

```
Uninstall[link]
-> /Users/kratky/Library/Mathematica/
    Applications/Sample.app
```

Finally, `Uninstall[link]` terminates the AppleScript applet and removes the Mathematica definitions set up by it.

Requesting Evaluations by the Kernel

The implementation of the AppleScript handler that has been installed as an external Mathematica function can itself request evaluations by the kernel between the time it is called and it returns its result. For example, you might want Mathematica to assist you in computing the result, or you might want to trigger some side effect such as displaying an error message in the Mathematica front end. The scripting addition command [MathLink evaluate](#) can be used for that purpose.

As an example, let's go back to the handler `addtwo` of the script object `MyHandler` and change the code so that it uses the Mathematica function `Plus` to calculate the result of the addition (the unchanged portions of the code have been omitted):

```
script MyHandler
...
on addtwo(a, b)
    return MathLink evaluate {class:expression, ↵
        head:a reference to |Plus|, ↵
        elements:{a, b}} on gLink
end addtwo
...
end script
```

Using multiple [MathLink evaluate](#) commands in succession on the link, the AppleScript handler can initiate dialogs of arbitrary length and complexity with the Mathematica kernel before it returns.

Error Handling

The `addtwo` handler is still missing an extremely important aspect of programming: error checking. If you do not care about error checking at all in your handler code, the scripting addition will simply return the error symbol `$Failed` to the kernel, whenever the execution of a handler returns an AppleScript error. An AppleScript error can be raised implicitly by the AppleScript interpreter or explicitly using the AppleScript `error` statement.

A better approach to error handling is to define a Mathematica message for each external function defined. A Mathematica message definition consists

of a message name and a message text. Here's an example of a message definition:

```
AddTwo::failed = "There has been an error `1`."
```

Each Mathematica symbol has a list of messages associated with it. Assignments for `s::tag` are stored in the message list of symbol `s`. After a message has been defined, it can be printed using the Mathematica function `Message`:

```
Message[AddTwo::failed, -128]
```

The argument `-128` will replace the placeholder ``1`` in the message text upon printing.

Let's add the error handling code to the script object `MyHandler` (the unchanged portions of the code have been omitted):

script `MyHandler`

```
...
property addtwo_Begin : {↵
    "AddTwo::usage = \"AddTwo[x, y] gives the sum of two machine
    integers x and y.\", ↵
    "AddTwo::failed = \"There has been an error `1`.\""}
on addtwo(a, b)
    try
        return a + b
    on error number errNum
        MathLink evaluate "Message[AddTwo::failed," & errNum & "]" ↵
        on gLink
        error -- propagate error -> results in $Failed
    end try
end addtwo
...
end script
```

The single string value of `MyHandler`'s property `addtwo_Begin` has been replaced by a list of strings, each of which will be evaluated before the extern function `AddTwo` is defined. The additional string is the definition of the error message `AddTwo::failed`.

The whole body of the `addtwo` handler has been wrapped by a `try` statement. If an error occurs, the error handler of the `try` statement triggers the Mathematica message by executing MathLink evaluate. The error is then propagated by calling **error**, which results in the symbol `$Failed` being sent to the kernel.

Now let's see the error messages in action. In the Mathematica notebook enter the following expression:

```
AddTwo[2^32, 2^32]
->AddTwo::failed: There has been an error -1700.
   $Failed
```

The value 2^{32} is too large to be representable as an AppleScript integer and will thus be passed as an AppleScript record of type [expression](#) to the AppleScript handler `addtwo`. Because addition is not supported by the AppleScript record type, the statement `return a + b` will make the AppleScript interpreter throw the error number -1700.

2.6 Using a Remote Mathematica Kernel

Because MathLink is a platform-independent communication protocol, you can use the MathLink for AppleScript to communicate with a remote Mathematica kernel on any platform supported by Mathematica. If the Parallel Computing Toolkit is installed, calculations can also be carried out on remote computing clusters driven by gridMathematica.

The task of using a remote Mathematica kernel can be broken down into the following sub-tasks:

- Launch: the Mathematica kernel needs to be launched on the remote machine.
- Connect: a connection needs to be established between the remote kernel and the local AppleScript program.
- Calculate: perform calculations with the kernel.
- Quit: quit Mathematica kernel on the remote machine.

The solution of the first two tasks depends on the platform that the Mathematica kernel is running on and on the network infrastructure between the communicating machines. Once the connection has been established, the last two tasks are trivial and do not differ from using a local Mathematica kernel. The following sections present three different methods for using a remote Mathematica kernel.

Launch and Connect via Remote Apple Events

If you want to target a remote Mathematica kernel running on another Mac OS X machine, the easiest solution is to use remote Apple Events. Apple Events are the mechanism that the AppleScript interpreter uses to send messages to applications targeted via the `tell application` AppleScript command. This mechanism works both for local and remote applications. Remote Apple Events are sent over the net using a proprietary Apple protocol called *eppc*.

To make the remote Mac OS X machine running the Mathematica kernel accessible for remote Apple Events, you must first set up the network access to the server machine accordingly. In the System Preferences application, select the Panel Sharing and switch to the Services tab. In the list of services select Remote Apple Events and press the Start button.

Because the remote machine will now handle MathLink AppleScript commands, the MathLink Scripting addition needs to be installed on the remote machine (see page 5).

The following AppleScript shows how to evaluate a simple expression with the Mathematica kernel on the remote machine:

```
tell application "Finder" of machine "eppc://192.168.1.51"  
  try  
    «event ascrdnt» -- force loading of MathLink OSAX  
  end try  
  set link to MathLink launch  
    "/"Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"  
    & space & "-mathlink"  
  MathLink evaluate "1+2" on link  
  MathLink close link  
end tell
```

When the script is run in the Script Editor application, the AppleScript interpreter will send the commands enclosed by **tell application** command to the Finder on the machine with the IP address 192.168.1.51. When the first command is sent to the remote Finder you will be prompted a password. All users that have an account on the remote machine are allowed to connect using either their login name or full name.

Remote Apple Events always need to be sent to a particular application aware of handling Apple Events. Because the Mathematica kernel does not support Apple Events itself, we will use the Finder as host application for handling MathLink related AppleScript commands.

The cryptic command «[event](#) ascrdnt» is needed to force the Finder to load the MathLink scripting addition into its application context. If this command is omitted, the Finder will not be able to handle MathLink related commands and will raise error number -1708 ("message cannot be handled") upon receiving such a command (see Apple Developer Technical Q&A QA1070).

The command [MathLink launch](#) will launch the Mathematica kernel as a sub process of the Finder on the remote machine. The full path to the Mathematica kernel on the remote machine's file system must be specified upon launch. The MathLink scripting addition installed on the remote machine will establish a local MathLink connection to the kernel.

[MathLink evaluate](#) sends a textual expression to the Mathematica kernel for evaluation via the MathLink connection. The final statement [MathLink close](#) closes the MathLink connection and quits the Mathematica kernel process on the remote machine.

It should be noted that the remote Finder is used as host for the MathLink scripting addition, which acts as a *local* client to the Mathematica kernel in this scenario. The transport of requests and results over the network is

handled entirely by AppleScript via remote Apple Events and is completely invisible to the Mathematica kernel.

Launch via Remote Apple Events and Connect via MathLink

This section presents another solution for targeting a remote kernel running on a Mac OS X machine. This solution does not require the MathLink Scripting addition to be installed on the remote machine, because remote Apple Events are only used to launch the Mathematica kernel. The local AppleScript program talks to the remote Mathematica kernel directly via MathLink:

```

set link to MathLink create with protocol "TCPIP"
set linkName to first item of elements of link
tell application "Finder" of machine "eppc://192.168.1.51"
do shell script
"/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
& space & "-mathlink -linkmode connect -linkprotocol TCPIP"
& space & "-linkname" & space & linkName
& space & "> /dev/null 2>&1 &"
end tell
MathLink activate link -- wait for kernel to connect
MathLink evaluate "1+2" on link
MathLink close link

```

When the script is run in the Script Editor application, the command [MathLink create](#) creates a TCPIP based MathLink link for the remote Mathematica kernel to connect to. The command picks an unused port number on the default network interface and returns an [expression](#) object with the name of the created link (e.g. "49857@192.168.1.56") as first element.

The script launches the Mathematica kernel on the remote machine with the IP address 192.168.1.51 as background process by executing the built-in AppleScript command [do shell script](#). The full path to the Mathematica kernel on the remote machine's file system must be specified upon launch. On the remote side, the command line options `-linkmode connect -linkprotocol TCPIP -linkname port@host` instruct the kernel to connect to the TCPIP link on the local machine we created earlier with [MathLink create](#). The output redirection options `> /dev/null 2>&1 &` ensure that [do shell script](#) returns immediately and that the kernel is launched as a background process (see Apple Developer Technical Note TN2065).

The AppleScript then executes [MathLink activate](#) on the link. This blocks the script until the remote kernel connects to the link. Once the MathLink connection has been established, sending expressions to the remote kernel with [MathLink evaluate](#) works the same way as for a local kernel. Closing the link with [MathLink close](#) on the client side will also quit the Mathematica kernel on the remote machine.

Launch via Secure Shell and Connect via MathLink

The most portable solution for using a remote Mathematica kernel is to launch the kernel via secure shell (ssh). Secure shell is a program for logging into a remote machine and for executing commands on a remote machine. It provides secure encrypted communications between two hosts over an insecure network. Secure shell is preconfigured on most UNIX-based systems out of the box and can be installed on other platforms like Microsoft Windows using third party tools.

A secure shell daemon (sshd) needs to be running on the remote machine where the Mathematica kernel resides. You also have to enable key-based (passwordless) authentication by adding the local machine's ssh public key to the remote machine's ssh authorized keys. There are plenty tutorials available on the web on setting up key-based ssh logins by googling for "passwordless ssh". For instructions on setting up sshd under Windows, see <http://support.wolfram.com/gridmathematica/ssh/windowsinstall.html>

On the local machine we use the ssh client tool `/usr/bin/ssh` to make a connection to the secure shell daemon on the remote machine and to launch the Mathematica kernel. The necessary command can be sketched as follows:

```
/usr/bin/ssh remote host "launch mathematica kernel"
```

In the previous section we have already seen how to launch the Mathematica kernel. By leveraging this knowledge, we can write the AppleScript using `/usr/bin/ssh` in the following way:

```
set link to MathLink create with protocol "TCPIP"
set linkName to first item of elements of link
do shell script
  "/usr/bin/ssh 192.168.1.51" & space
  & "\""
  & "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
  & space & "-mathlink -linkmode connect -linkprotocol TCPIP"
  & space & "-linkname" & space & linkName
  & space & "> /dev/null 2>&1 &"
  & "\""
MathLink activate link -- wait for kernel to connect
MathLink evaluate "1+2" on link
MathLink close link
```

Once the TCPIP based MathLink link has been created for the remote Mathematica kernel to connect to, the script uses [do shell script](#) to establish a secure shell connection to the remote machine 192.168.1.51 and executes the Mathematica kernel there. The command line options `-linkmode connect -linkprotocol TCPIP -linkname port@host` tell the kernel to connect to the TCPIP link we created earlier with [MathLink create](#). The output redirection options `> /dev/null 2>&1 &` ensure that the kernel is launched as a background process.

The format of the kernel path used in the [do shell script](#) command is platform dependent. The path shown in the sample script is valid for the default installation location of Mathematica on Mac OS X only. If the remote machine runs UNIX, `/usr/local/bin/math` should be used instead.

If you are connecting to an older version of the Mathematica kernel, which does not support the TCPIP protocol, you can use the older, less efficient TCP protocol instead. Specify [with protocol](#) "TCP" upon creating the link on the local side and `-linkprotocol TCP` upon launching the kernel on the remote side.

Further communication between the script and the Mathematica kernel is conducted through the use of MathLink and is identical to the scenario described in the previous section.

3 Command Reference

3.1 Overview

MathLink for AppleScript offers the following AppleScript commands for managing MathLink connections:

Command	Description
MathLink create	Create a MathLink link for another program to connect to.
MathLink connect	Connect to a MathLink link created by another program.
MathLink close	Close a MathLink connection.
MathLink launch	Start an external program and open a MathLink connection to it.
MathLink activate	Activate a MathLink connection.
MathLink links	Return list of MathLink connections currently open.

Once a MathLink connection has been established, the following commands can be used to exchange data over the connection:

Command	Description
MathLink read	Read one expression from the specified MathLink connection.
MathLink write	Write an expression to the specified MathLink connection.
MathLink peek	Peek ahead one expression on the specified MathLink connection without consuming it.
MathLink ready	Test whether there is an expression ready to be read from the specified MathLink connection.
MathLink evaluate	Evaluate an expression as Mathematica input.
MathLink enter	Enter an expression as Mathematica input line.
MathLink abort	Sends an abort message to the program at the other end of the specified MathLink connection.
MathLink interrupt	Sends an interrupt message to the program at the other end of the specified MathLink connection.
MathLink terminate	Sends a terminate message to the program at the other end of the specified MathLink connection.

To facilitate callbacks from the Mathematica kernel, the following commands are available:

Command	Description
MathLink install	Install AppleScript handlers as Mathematica functions.
MathLink handle	Handle a callback from Mathematica on a MathLink connection.

3.2 MathLink create

The [MathLink create](#) command syntax:

```

MathLink create [string] -- The name of the connection to create.
  [with protocol string] -- The underlying data transport protocol ('TCP' or 'TCPIP').
  [with options string]  -- The set of desired options (e.g., 'MLDontInteract').
  [with host string]     -- The host name or IP number to create link for.
Result: expression      -- The MathLink connection.

```

[MathLink create](#) creates a MathLink link with the specified name for another program to connect to. The name of the connection is a TCP port number (a positive integer such as 2300), specified as a string. If your system has more than one network interface, the TCPIP protocol allows the user to specify the desired Ethernet interface by giving the name in the format *port@host*, where *host* is the IP number of the desired interface. If you omit the direct parameter or specify an empty string, [MathLink create](#) picks an unused port number on the default network interface.

The parameter [with protocol](#) lets you specify the underlying network protocol to use. Valid options are 'TCP' and 'TCPIP'. 'TCPIP' is a newer protocol and runs significantly faster than its predecessor 'TCP'. The 'TCP' protocol is still supported to enable communication with older versions of the MathLink library. If you omit the parameter, 'TCPIP' will be used.

To have the [MathLink create](#) command pick its own ports, but to constrain the link to a certain network interface, specify the interface's IP number in the [with host](#) parameter as a string. The given string will be passed to the MathLink library function MLOpenArgv with the -linkhost argument, when the link is created.

Using the parameter [with options](#) you can specify options that will be passed to the MathLink library function MLOpenArgv with the -linkoptions argument, when the link is created.

The command returns an object of type [expression](#), whose head is the symbol LinkObject. This object represents the created link and remains valid until the link is closed.

[MathLink create](#) returns immediately. The underlying connection is activated, when the first expression is written to or read from the link.

The following example creates a link with a default port name on the default network interface and default options. The actual name of the created link can be obtained by accessing the first item of the [elements](#) property of the returned expression.

```
MathLink create
->{
  class:expression,
  head:|LinkObject|,
  elements:{
    "55678@192.168.1.51,55679@192.168.1.51",
    1
  }
}
```

This example shows how to create a link at port 3000 that uses the older version of the MathLink protocol ('TCP'):

```
MathLink create "3000" with protocol "TCP"
->{
```

```

class:expression,
head:|LinkObject|,
elements:{
  "3000@Friaul.local",
  4
}
}

```

Please note that the [MathLink create](#) command may fail, if the specified port is already in use by another process. If you let the command choose a default port name, an unused port will be used.

3.3 MathLink connect

The [MathLink connect](#) command syntax:

```

MathLink connect [string] -- The name of the link to connect to.
                                -- If omitted, connect to parent process.
\[with protocol string] -- The underlying data transport protocol
                                -- ('TCP' or 'TCPIP').
\[with options string] -- Options.
Result: expression -- The MathLink connection.

```

[MathLink connect](#) connects to a MathLink link created by another program on the same machine or on a remote computer system. In combination with [MathLink create](#) the command can be used to set up a peer-to-peer connection between two processes. The link to connect to must have been created by calling `LinkCreate` in a Mathematica session, by calling [MathLink create](#) in an AppleScript, or by using an appropriate MathLink library function (`MLOpenArgv`, `MLOpenString`).

The link name must be specified in the format *port@host*. Ports are typically specified by numbers. If you omit the link name the AppleScript command attempts to establish a MathLink connection to the parent process.

The parameter [with protocol](#) lets you specify the underlying network protocol to use. Valid options are 'TCP' and 'TCPIP'. 'TCPIP' is a newer protocol and runs significantly faster than its predecessor 'TCP'. The 'TCP' protocol is still supported to enable communication with older versions of the MathLink library. The protocol specified must match the protocol specified at the other side of the link. If you omit the parameter 'TCPIP' will be used.

Using the parameter [with options](#) you can specify options that will be passed to the MathLink library function `MLOpenArgv` with the `-linkoptions` argument, when the link is connected.

The command returns an object of type [expression](#), whose head is the symbol `LinkObject`. This object represents the created link and remains valid until the link is closed.

[MathLink connect](#) returns immediately. The underlying connection is activated, when the first expression is written to or read from the link.

The following example shows how to connect to a link on port 3000 on the local machine (whose IP address is 192.168.1.51):

```
MathLink connect "3000"
--> {
  class:expression,
  head: |LinkObject|,
  elements:{"3000@192.168.1.51", 1}
}
```

Connecting to the remote machine 192.168.1.54 on port 3000:

```
MathLink connect "3000@192.168.1.54"
--> {
  class:expression,
  head: |LinkObject|,
  elements:{"3000@192.168.1.51", 1}
}
```

Connecting to the parent process:

```
MathLink connect
--> {
  class:expression,
  head: |LinkObject|,
  elements:{"ParentLink", 1}
}
```

3.4 MathLink close

The [MathLink close](#) command syntax:

[MathLink close](#) [expression](#) -- The connection to close.

[MathLink close](#) closes an open MathLink connection. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol [LinkObject](#), as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

Closing an open MathLink connection does not necessarily terminate the program at the other end of the link. Any data buffered in the link is sent before the link is closed.

3.5 MathLink activate

The [MathLink activate](#) command syntax:

[MathLink activate](#) [expression](#) -- The connection to activate.

[MathLink activate](#) activates a MathLink connection, waiting for the program at the other end to respond. The direct parameter of the command

must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

3.6 MathLink launch

The [MathLink launch](#) command syntax:

```
MathLink launch string    -- The external program to launch.
  [with protocol string] -- The underlying data transport protocol
                        -- ('TCP' or 'TCPIP').
  [with options string] -- Optional options (e.g., MLDontInteract).
Result: expression      -- The connection to the launched program.
```

[MathLink launch](#) starts the external program specified by the direct parameter and opens a MathLink connection to it. The program must be specified by its full POSIX path and will be run as child process to the calling process. The direct parameter may also include command line options to pass thru to the launched program.

The parameter [with protocol](#) lets you specify the underlying network protocol to use. Valid options are 'TCP' and 'TCPIP'. 'TCPIP' is a newer protocol and runs significantly faster than its predecessor 'TCP'. The 'TCP' protocol is still supported to enable communication with older versions of the MathLink library.

Using the parameter [with options](#) you can specify options that will be passed to the MathLink library function `MLOpenArgv` with the `-linkoptions` argument, when the link is created.

The command returns an object of type [expression](#), whose head is the symbol `LinkObject`. This object represents the created link and remains valid until the link is closed.

The following example launches the Mathematica kernel in MathLink mode:

```
MathLink launch
  "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
  -mathlink"
->{
  class:expression,
  head: |LinkObject|,
  elements:{
    "/Applications/Mathematica 5.2.app/Contents/MacOS/MathKernel"
    -mathlink", 8}
  }
```

3.7 MathLink links

The [MathLink links](#) command syntax:

MathLink links

[with name string] -- List only links with the specified name.
Result: a list of expression -- The list of MathLink connections.

MathLink links gives a list of all MathLink connections that are currently open. It returns a list of objects of type expression, whose head is the symbol `LinkObject`.

If the with name parameter is present, the command only lists the links with the specified name.

Example:

MathLink links

```
--> {
    {
        class:expression,
        head:|LinkObject|,
        elements:{
            "61310@192.168.1.51,61311@192.168.1.51",
            1
        }
    },
    {
        class:expression,
        head:|LinkObject|,
        elements:{
            "61318@Friaul.local",
            2
        }
    }
}
```

3.8 MathLink read

The MathLink read command syntax:

MathLink read expression -- The connection to read from.
[as type class] -- The form in which to read and return data.
Result: anything -- One expression read from the connection.

MathLink read reads one expression from the specified MathLink connection. The direct parameter of the command must be an active object of type expression, whose head is the symbol `LinkObject`, as returned by the commands MathLink create, MathLink open or MathLink launch.

MathLink read will block until it has read a complete expression or the command's time out limit has expired. You can test whether an expression is ready to be read from the link by calling MathLink ready. A **with timeout** statement lets you change how long the command waits. AppleScript's default time out is one minute.

If the user cancels the command prematurely by typing Command-period or the time out limit has expired, the command skips to the end of the expression on the link and returns an error.

The optional parameter [as](#) lets you specify the form in which to read and return the data read from the link. If the data to be read is not a valid value for the specified type, the [MathLink read](#) command returns an error. If the [as](#) parameter is omitted, the command chooses an appropriate AppleScript type for the result or falls back to the record type [expression](#).

3.9 MathLink write

The [MathLink write](#) command syntax:

[MathLink write](#) [anything](#) -- The expression to write to the specified connection.
 [to](#) [expression](#) -- The connection to write to.

[MathLink write](#) writes one expression to the specified MathLink connection. The parameter [to](#) of the command must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

The head of the expression written will often be a packet, which specifies how the expression should be processed by the program that receives it. When you use [MathLink write](#) to send data to the Mathematica kernel, one of the packet types [evaluatepkt](#), [enterexprpkt](#), or [entertextpkt](#) should be used.

[MathLink write](#) will block until the complete expression has been written via the underlying communication mechanism or the command's time out limit has expired. A **with timeout** statement lets you change how long the command waits. AppleScript's default time out is one minute. The user can cancel the command prematurely by typing Command-period.

3.10 MathLink peek

The [MathLink peek](#) command syntax:

[MathLink peek](#) [expression](#) -- The connection to read from.
 [[as](#) [type class](#)] -- The form in which to read and return data.
 [[head only](#) [boolean](#)] -- Only return the head of the expression.
 Result: [anything](#) -- One expression read from the connection.

[MathLink peek](#) reads one expression from the link, and then resets the link to the state prior to reading the expression. You can use this command to "peek ahead" without consuming anything off the link. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink peek](#) will block until it has read a complete expression or the command's time out limit has expired. You can test whether an expression is ready to be read from the link by calling [MathLink ready](#). A **with timeout** statement lets you change how long the command waits. AppleScript's default time out is one minute.

If the user cancels the command prematurely by typing Command-period or the time out limit has expired, the command skips to the end of the expression on the link and returns an error.

If the parameter [head only](#) is set to [true](#), the command only returns the head of the next expression on the link. This is handy for inspecting the packet type of the next expression waiting to be read from the link.

The optional parameter [as](#) lets you specify the form in which to read and return the data read from the link. If the data to be read is not a valid value for the specified type, the [MathLink peek](#) command returns an error. If the [as](#) parameter is omitted, the command chooses an appropriate AppleScript type for the result or falls back to the record type [expression](#).

3.11 MathLink ready

The [MathLink ready](#) command syntax:

MathLink ready expression	-- The connection to test.
Result: boolean	-- True if it expression available, false if not.

[MathLink ready](#) tests whether there is an expression ready to read from the specified MathLink connection. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol LinkObject, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink ready](#) will always return immediately and will not block. If [MathLink ready](#) returns [true](#), then a subsequent [MathLink read](#) command will not block under normal circumstances. If [MathLink ready](#) returns [false](#), then [MathLink read](#) will block, and will not return until something becomes available to read on the link. [MathLink ready](#) only tests whether there is any data to read; it cannot determine whether the data represents a complete expression.

3.12 MathLink evaluate

The [MathLink evaluate](#) command syntax:

MathLink evaluate anything	-- The expression to evaluate on the specified connection.
on expression	-- The connection to evaluate expression on.
[as type class]	-- The form in which to return evaluation result.
Result: anything	-- Result of evaluation.

[MathLink evaluate](#) is a convenience command, which evaluates an expression on the specified link and returns the result of evaluation. The command assumes that the program on the other side is the Mathematica kernel.

The parameter [on](#) of the command must be an active object of type [expression](#), whose head is the symbol LinkObject, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

The optional parameter [as](#) lets you specify the form in which to return the result of the evaluation. If the data to be read is not a valid value for the specified type, [MathLink evaluate](#) returns an error. If the [as](#) parameter is omitted, the command chooses an appropriate AppleScript type for the result or falls back to the record type [expression](#).

The execution of the command consists of the following steps:

1. If the direct parameter is a string, it is embedded into an implicit ToExpression function, which ensures that the string is interpreted as Mathematica input.
2. The direct parameter is wrapped into a packet with the head [evaluatepkt](#).
3. The [evaluatepkt](#) expression is written to the Mathematica kernel.
4. The command waits for the Mathematica kernel to send back a [returnpkt](#) expression that holds the result of the evaluation.
5. The result is extracted from the [returnpkt](#), cast to the result type (if specified) and returned.

[MathLink evaluate](#) will block until it receives a [returnpkt](#) expression from the Mathematica kernel or the command's time out limit has expired. A **with timeout** statement lets you change how long the command waits for an answer. AppleScript's default time out is one minute. The user can cancel the command prematurely by typing Command-period.

The following example shows that evaluating the ListPlot function returns a Graphics expression:

```
MathLink evaluate "ListPlot[Table[n, {n, 5}]]" on link
->{class:expression, head:Graphics, elements:{{1, 1}, {2, 2}, {3, 3},
{4, 4}, {5, 5}}, {PlotRange:Automatic,
|AspectRatio|:{class:expression, head:Power, elements:{GoldenRatio,
-1}}, |DisplayFunction|:$DisplayFunction, |ColorOutput|:Automatic,
|Axes|:Automatic, |AxesOrigin|:Automatic, |PlotLabel|:None,
|AxesLabel|:None, |Ticks|:Automatic, |GridLines|:None, |Prolog|:{},
|Epilog|:{}, |AxesStyle|:Automatic, |Background|:Automatic,
|DefaultColor|:Automatic, |DefaultFont|:$DefaultFont,
|RotateLabel|:true, |Frame|:false, |FrameStyle|:Automatic,
|FrameTicks|:Automatic, |FrameLabel|:None, |PlotRegion|:Automatic,
|ImageSize|:Automatic, |TextStyle|:$TextStyle,
|FormatType|:$FormatType}}}
```

3.13 MathLink enter

The [MathLink enter](#) command syntax:

MathLink enter anything	-- The expression to enter on the specified connection.
on expression	-- The connection to enter expression on.
[as type class]	-- The form in which to return evaluation result.
Result: anything	-- Result of entering expression.

[MathLink enter](#) is a convenience command, which enters an expression on the specified link corresponding to an input line and returns the result corresponding to an output line. The command assumes that the program on the other side is the Mathematica kernel.

The parameter [on](#) of the command must be an active object of type [expression](#), whose head is the symbol LinkObject, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

The optional parameter [as](#) lets you specify the form in which to return the result of the evaluation. If the data to be read is not a valid value for the specified type, the [MathLink enter](#) command returns an error. If the [as](#) parameter is omitted, the command chooses an appropriate AppleScript type for the result or falls back to the record type [expression](#).

The execution of the command consists of the following steps:

1. If the direct parameter is a string, it is wrapped into an [entertextpkt](#) packet. Otherwise the direct parameter is wrapped into an [enterexprpkt](#) packet.
2. The packet expression is written to the Mathematica kernel.
3. The command waits for the Mathematica kernel to send back a [returntextpkt](#) or [returnexprpkt](#) (4) or a series of [displaypkt](#) objects followed by a [displayendpkt](#) (5) or a [syntaxpkt](#) preceded by a [textpkt](#) containing an error message (6).
4. The result is extracted from the [returntextpkt](#) or [returnexprpkt](#), cast to the result type (if specified) and returned.
5. PostScript data contained in the [displaypkts](#) and the [displayendpkt](#) is concatenated, cast to the result type (if specified) and returned.
6. The error information is extracted from both the [textpkt](#) and the [syntaxpkt](#) and returned as an AppleScript error message.

[MathLink enter](#) will block until the whole answer has been received from the Mathematica kernel or the command's time out limit has expired. A **with timeout** statement lets you change how long the command waits for an answer. AppleScript's default time out is one minute. The user can cancel the command prematurely by typing Command-period.

The following example shows that entering the `ListPlot` function corresponding to an input line returns PostScript data, which represents the rendered list plot:

```
MathLink enter "ListPlot[Table[n, {n, 5}]]" on link
-> "%%Creator: Mathematica
%%AspectRatio: .61803
MathPictureStart
/Mabs {
Mgmatrix idtransform
Mtmatrix dtransform
} bind def
...
.7381 .45617 Mdot
.97619 .60332 Mdot
% End of Graphics
MathPictureEnd
```

Entering syntactically incorrect input result in an AppleScript error message (Note the missing bracket at then end of the expression):

```
MathLink enter "ListPlot[Table[n, {n, 5}]]" on link
-> Syntax::sntxi: Incomplete expression; more input is needed.
```

3.14 MathLink abort

The [MathLink abort](#) command syntax:

MathLink abort expression	-- The connection to interrupt.
[waiting for response boolean]	-- wait for response from program on other end
	-- of link before returning (default is false)
Result: anything	-- Response to abort message.

[MathLink abort](#) sends a (low level) abort message to the program at the other end of the specified connection. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink abort](#) is identical to what happens in the notebook front end when you choose the Abort Evaluation menu command from the Kernel menu. Mathematica responds to this command by terminating the current evaluation and returning the symbol `$Aborted`.

Sometimes the kernel is in a state where it cannot respond immediately to an abort message. The optional Boolean parameter [waiting for response](#) makes the command wait for an answer from the program on the other end of the link. If the parameter is omitted, the call will return immediately and will not block.

You can use the command to abort time-consuming calculations as in the following example:

```

try
  with timeout of 3 seconds
    MathLink evaluate "FactorInteger[2^777-1]" on link
  end timeout
on error errMsg number errNum
  if errNum = -1712 then -- -1712 means time out occurred
    MathLink abort link with waiting for response
  end if
end try

```

The [MathLink evaluate](#) call is wrapped in a **with timeout** statement which makes the AppleScript interpreter raise the error -1712 (errAETimeout), if the command does not complete within the specified time (3 seconds). The error handler aborts the evaluation of FactorInteger and waits for the kernel to return \$Aborted.

3.15 MathLink interrupt

The [MathLink interrupt](#) command syntax:

```

MathLink interrupt expression      -- The connection to interrupt.
[waiting for response boolean]    -- wait for response from program on other end
                                   -- of link before returning (default is false)
Result: anything                  -- Response to interrupt message.

```

[MathLink interrupt](#) sends a (low level) interrupt message to the program at the other end of the specified connection. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol LinkObject, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink interrupt](#) is identical to what happens in the notebook front end when you choose the Interrupt Evaluation menu command from the Kernel menu. Mathematica responds to this command by interrupting the current evaluation and sending back a [menupkt](#) that contains choices for what to do next. The choices can depend on what the kernel is doing at the moment, but in most cases they include aborting, continuing, or entering a dialog. It is not likely that you will want to have to deal with this list of choices on your own, so you might choose instead to call [MathLink abort](#) and just stop the computation.

Sometimes the kernel is in a state where it cannot respond immediately to an interrupt message. The optional Boolean parameter [waiting for response](#) makes the command wait for an answer from the program on the other end of the link. If the parameter is omitted, the call will return immediately and will not block.

You can use the command to interrupt time-consuming calculations as in the following example:

```

MathLink write {class:expression, head:entertextpkt,
  elements:{"FactorInteger[2^777-1]"}} to link

```

```

MathLink ready link
-> false
MathLink interrupt link
MathLink read link
-> {class:expression, head:menupkt, elements:{1, "Interrupt>"}}
MathLink write "a" to link -- choose "abort" option
MathLink read link -- discard $Abort on link
-> {class:expression, head:returnpkt, elements:{|$Aborted|}}

```

3.16 MathLink terminate

The [MathLink terminate](#) command syntax:

[MathLink terminate](#) expression -- The connection to interrupt.

[MathLink terminate](#) sends an interrupt request to the program at the other end of the specified connection. The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol LinkObject, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink terminate](#) sends a terminate request to Mathematica. This is the strongest step you can take to tell the kernel to shut down, short of killing the kernel process with operating system commands. In "normal" operation of the kernel, when you call [MathLink close](#) on the link, the Mathematica kernel will quit. In some cases, however, generally only if the kernel is currently busy computing, it will not quit. In such cases you can force the kernel to quit immediately by calling [MathLink terminate](#). You should always call [MathLink close](#) immediately afterward:

```

try
  with timeout of 3 seconds
    MathLink evaluate "FactorInteger[2^777-1]" on link
  end timeout
on error
  MathLink terminate link -- terminate kernel on error
end try
MathLink close link

```

3.17 MathLink install

The [MathLink install](#) command syntax:

MathLink install script	-- The script containing MathLink callbacks.
on expression	-- The connection to install AppleScript on.
Result: a list of string	-- The names of the AppleScript handlers installed.

[MathLink install](#) installs handlers defined in a script object as external Mathematica functions on the specified MathLink connection. The command assumes that the program on the other side is the Mathematica

kernel and that the kernel is currently executing the function `Install[link]` and awaits definitions on the specified link.

The parameter [on](#) of the command must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

The script object containing handlers is specified as direct parameter of the command. [MathLink install](#) uses AppleScript's reflection API to look for suitable handlers defined in the script object. The recommended structure of the script object is sketched in the following AppleScript code:

```
script my_script
  -- initialization
  property Evaluate_Begin : {"BeginPackage["packagename`\""]"}
  -- functions to install
  property foo_Begin : {""}
  property foo_Pattern : "FunctionPattern[arg1, arg2, arg3]"
  property foo_Arguments : "{arg1, arg2, arg3}"
  on foo(x, y, z)
    return ""
  end foo
  property foo_End : {""}
  -- termination
  property Evaluate_End : {"EndPackage[]"}
end script
```

Only the properties `foo_Pattern` and `foo_Arguments` are required for the handler named `foo` to be installable as a Mathematica function. The following table describes the purpose of each property:

Property name	Description
<code>Evaluate_Begin</code>	Optional Mathematica input to evaluate, <i>before</i> installing any handlers. The property value can be a string or a list of strings. Each string must contain a separate Mathematica statement or definition.
<code>foo_Begin</code>	Optional Mathematica input to evaluate, <i>before</i> installing the handler named <code>foo</code> . The property value can be a string or a list of strings. Each string must contain a separate Mathematica statement or definition.
<code>foo_Pattern</code>	The pattern to be defined to call the handler <code>foo</code> . This will become the left-hand side of a function definition, exactly as you would type it if you were creating the entire function in Mathematica.
<code>foo_Arguments</code>	The arguments to the handler <code>foo</code> as Mathematica code. The number of argument expressions must correspond to the number of positional parameters declared in the AppleScript handler.
<code>foo_End</code>	Optional Mathematica input to evaluate, <i>after</i> the handler named <code>foo</code> has been installed. The property value can be a string or a list of strings. Each string must contain a separate Mathematica statement or definition.
<code>Evaluate_End</code>	Optional Mathematica input to evaluate, <i>after</i> all handlers

	have been installed. The property value can be a string or a list of strings. Each string must contain a separate Mathematica statement or definition.
--	--

[MathLink install](#) returns a list of names of the handlers that have been installed as external Mathematica functions. The script object must at least contain one handler suitable for installation for the command to succeed.

[MathLink install](#) sets up definitions in the Mathematica kernel, which send [callpkt](#) objects over the link whenever external functions are called.

You can remove the definitions later by calling `Uninstall[link]` later from the Mathematica side of the link.

3.18 MathLink handle

The [MathLink handle](#) command syntax:

[MathLink handle](#) [expression](#) -- The connection to handle callback on.

[MathLink handle](#) waits for a [callpkt](#) object on the specified MathLink connection. The [callpkt](#) contains a numeric identifier for the AppleScript handler to call and a list of arguments. The [callpkt](#) object is handled by translating it into an AppleScript handler call of the specified script handler. The result returned by the AppleScript handler is wrapped into a [returnpkt](#) object and sent back over the link.

The direct parameter of the command must be an active object of type [expression](#), whose head is the symbol `LinkObject`, as returned by the commands [MathLink create](#), [MathLink open](#) or [MathLink launch](#).

[MathLink handle](#) will block until it has received a [callpkt](#) object or the command's time out limit has expired. You can test whether an expression is ready to be read from the link by calling [MathLink ready](#). A **with timeout** statement lets you change how long the command waits. AppleScript's default time out is one minute.

The primary use of the [MathLink handle](#) command is polling in the [idle](#) handler of an AppleScript applet:

```
on idle
  try
    if MathLink ready gLink then
      MathLink handle gLink
    end if
    on error errStr number errNum
      if errNum > 0 then -- MathLink related error
        quit
      end if
    end try
    return 0.2 -- call idle handler again in 0.2 seconds
end idle
```