

CyberCron

COLLABORATORS

	<i>TITLE :</i> CyberCron		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		November 11, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	CyberCron	1
1.1	CyberCron documentation	1
1.2	Legal Mumbo Jumbo	1
1.3	Supplied Files	2
1.4	What is a cron?	2
1.5	Why another cron for the Amiga?	2
1.6	Starting CyberCron	3
1.7	The »CronTab« command line argument	3
1.8	The »LogFile« command line argument	4
1.9	The »SendMail« command line argument	4
1.10	The »DefStack« command line argument	5
1.11	The »DefPri« command line argument	5
1.12	The »CronPri« command line argument	5
1.13	The »PortName« command line argument	6
1.14	Creating a crontab file	6
1.15	The »:NAME« event line option	7
1.16	The input redirection event line option	8
1.17	The output redirection event line option	8
1.18	The error redirection event line option	8
1.19	The »:MAILUSER« event line option	8
1.20	The »:EXECONCE« event line option	9
1.21	The »:OBEYQUEUE« event line option	9
1.22	The »:NOLOG« event line option	9
1.23	The »:REXX« event line option	10
1.24	The »:STACK« event line option	10
1.25	The »:PRI« event line option	10
1.26	The »:CUSTOMSH« event line option	10
1.27	The »:SYSSH« event line option	11
1.28	CyberCron as an ARexx Function Host	11
1.29	The »SHUTDOWN« ARexx command	12

1.30 The »VERSION« ARexx command	12
1.31 The »SUSPEND« ARexx command	12
1.32 The »RESUME« ARexx command	13
1.33 The »NEW_EVENT_FILE« ARexx command	13
1.34 The »CLOSE_EVENT_FILE« ARexx command	13
1.35 The »NEW_LOG_FILE« ARexx command	13
1.36 The »CLOSE_LOG_FILE« ARexx command	14
1.37 The »SHOW_STATUS« ARexx command	14
1.38 The »PURGE_REXX_EVENTS« ARexx command	14
1.39 The »ADD_EVENT« ARexx command	15
1.40 The »DELETE_REXX_EVENT« ARexx command	15
1.41 The »DELETE_EVENT« ARexx command	15
1.42 The »LIST_EVENTS« ARexx command	15
1.43 The »SHOW_EVENT« ARexx command	16
1.44 The »SET_QUEUE_MAX« ARexx command	16
1.45 The »GET_QUEUE_MAX« ARexx command	17
1.46 The »EVENT_NEXT_EXEC« ARexx command	17
1.47 The »NEXT_EVENT_TO_EXEC« ARexx command	17
1.48 The »EXPAND_SSSC« ARexx command	18
1.49 The »SSSC_TO_ASCII« ARexx command	18
1.50 How job queues work in CyberCron	19
1.51 The internal naming scheme used by CyberCron	19
1.52 How to make CyberCron speak your language	20
1.53 The history of CyberCron	21
1.54 Changes in version 1.2 of CyberCron	21
1.55 Changes in version 1.3 of CyberCron	21
1.56 Changes in version 1.4 of CyberCron	22
1.57 Credits	23
1.58 Contacting the author	24
1.59 The use of »OPTIONS RESULTS« in an ARexx script	25
1.60 What file notification does under KickStart 2.04 or greater	25
1.61 NazCron	25
1.62 The »at« command	26
1.63 The »Batch« command	26
1.64 The Commodore PIPE: device	26
1.65 AMIGA User Interface Style Guide	26
1.66 System Creation	27

Chapter 1

CyberCron

1.1 CyberCron documentation

CyberCron

Copyright © 1992 by Christopher A. Wichura
All rights reserved.

Legal Mumbo Jumbo
Supplied Files
What is a cron?
Why another cron for the Amiga?
Starting CyberCron
Creating a crontab file
CyberCron as an ARexx Function Host
How job queues work in CyberCron
The internal naming scheme used by CyberCron
How to make CyberCron speak your language
The revision history of CyberCron
Credits
Contacting the author

1.2 Legal Mumbo Jumbo

CyberCron is not in the public domain. All source files, along with the resulting executable, are copyright by C. Wichura. You may not sell CyberCron. The only allowed charge that may be placed on CyberCron is for media and/or mailing costs.

CyberCron may be freely redistributed via BBSs, InterNet/Usenet, and disk libraries such as Fred Fish's, as long as the archive is not modified. Disk magazines and services that charge extra for file transfers may not distribute CyberCron.

In using CyberCron, you accept the responsibility for any damage or loss of productivity/money that may occur through or during its use. C. Wichura is not and cannot be held accountable.

1.3 Supplied Files

All source to CyberCron is included in the Source/* directory. Please do not redistribute changes you make. Instead, send them to me for inclusion in the master distribution.

The s directory contains a sample crontab file. This is the one I am currently using on my system. It contains a number of comments that give some helpful hints on how to use some of the more advanced features of CyberCron.

In REXXScripts is CCShow.rexx, based on the NCShow.rexx script that came with NazCron. It will call LIST_EVENTS and if there are active events, it will loop through the list calling SHOW_EVENT to display each one.

Also in the rexx directory you will find the files at.rexx and batch.rexx. These are the files Loren Rittle sent me that implement the UNIX at and batch commands using CyberCron. They have since been updated by Graham Walter to be much more complete.

The Catalogs directory is where translation catalogs for languages that CyberCron has been converted to are found. If you are running under KickStart 2.1 and would like to use CyberCron in your language, copy the appropriate file into your LOCALE:Catalogs/<language> directory.

1.4 What is a cron?

A cron is a program that runs in the background (i.e., requires no user intervention) and executes commands at specific times, as defined in a crontab file. It allows the automation of jobs that one runs regularly.

For example, many maintenance tasks can be handled by a cron event. Incremental system backups to a streaming tape could be started at a time when one is generally away from the machine every day or every week, for example. Or you might have a cron entry run a script that logs onto BIX, CompuServe, or wherever and downloads all your mail and unread messages automatically during the night when rates are lower.

A cron can also be used as a means of reminding yourself to do things. You might add an event that runs an alarm clock program in the morning to wake you up, for example. Or you might have it throw a requester up telling you to go to bed when it is getting late. The possibilities are endless.

1.5 Why another cron for the Amiga?

There have been several crons for the Amiga already. These include AmiCron, dcron, TPTCron and NazCron, to name a few. So what's the need for yet another cron?

CyberCron was written to take advantage of the new features of KickStart 2.04 (or greater), and thus has many advanced options that previous crons don't. It's this new functionality that makes switching over to CyberCron

worthwhile.

CyberCron uses the new `System()` call in `dos.library` to launch events. This call is far superior to the `Execute()` call of KickStart 1.3 days which the other crons use. Through the use of the `System()` call, CyberCron can support options such as the stack size or priority to launch specific events at.

CyberCron also understands `ARexx`. It can start events as `ARexx` commands and/or scripts. It also has an `ARexx` port which can accept a number of `ARexx` commands to control it while running.

CyberCron supports localization. If you are running it on a machine with `locale.library` and have a translation catalog for your local language, CyberCron's messages will appear in your language instead of English.

Finally, CyberCron also uses file notification to determine when the crontab file has been changed. This eliminates the performance hit that previous Amiga crons have suffered from due to the fact that they had to check the crontab file for updates once a minute.

1.6 Starting CyberCron

You can start CyberCron from either the CLI or Workbench.

When CyberCron is started from the Workbench, it tries to clone a copy of the Workbench's CLI structure. What this means is that the command path that existed at the time LoadWB was run will be propagated on to commands CyberCron launches.

CyberCron understands the following arguments:

```
[ CRONTAB <filespec>] [ LOGFILE <filespec>]
[ DEFSTACK <Default Stack>] [ DEFPRI <Default Priority>]
[ SENDMAIL <"{command} {from switch} %s {name switch} *"%s*"">]
[ CRONPRI <Cron's Priority>] [ PORTNAME <ARexx Portname>]
```

All arguments are optional. If not specified, CyberCron will use an appropriate default. See the specific options' documentation for more information.

To use these options from a Workbench icon, simply make each tooltype be in the form of `<option>=<setting>`. For example, if you wanted to specify a logfile of `T:CronLog`, you would have a tooltype that looked like

```
LOGFILE=T:CronLog
```

1.7 The »CronTab« command line argument

This specifies the crontab file that CyberCron will try and read. If not specified, `s:CronTab` will be used as the default.

The crontab file may also be changed while CyberCron is running by using the NEW_EVENT_FILE ARexx command.

TEMPLATE
CRONTAB/K

EXAMPLE
CronTab cron:ctab

1.8 The »LogFile« command line argument

This indicates where CyberCron is to log activities. If no filespec is given, logging will be turned off.

The logfile may also be changed while CyberCron is running by using the NEW_LOG_FILE ARexx command.

TEMPLATE
LOGFILE/K

EXAMPLE
LogFile T:CronLog

1.9 The »SendMail« command line argument

CyberCron supports sending the output of a command it has launched to a user via e-mail. For this to work, however, it needs to know how to send mail on the local system. This command tells CyberCron how to do this.

If the sendmail option is not specified then entries which try to direct their output to e-mail will not be able to. File output redirection will be honored instead.

The argument for this command should be in the form of

```
"<sendmail command> <from switch> %s <realname switch> *"%s*""
```

For example, for AmigaUUCP's sendmail one would use

```
"UUCP:c/SendMail -f %s -R *"%s*""
```

For CyberCron to be able to send mail, the sendmail command must read the mail message from its standard input. This is because CyberCron sets up a pipe to redirect output into sendmail. This also means that you must have PIPE: mounted before mail can be sent.

The sendmail command must also expect the To: and Subject: fields to be in its input instead of on the command line, and must generate the Message-ID: field itself.

AmigaUUCP's SendMail command meets all the above criteria.

TEMPLATE

```
SENDMAIL/K
```

EXAMPLE

```
SendMail "UUCP:c/SendMail -f %s -R *"%s*"
```

1.10 The »DefStack« command line argument

This is the default stack size to launch executables with. Values less than 2048 will be rejected.

This argument is ignored when launching ARexx events.

If not specified, the stack size that CyberCron was started with will be used as the default.

TEMPLATE

```
DEFSTACK/K/N
```

EXAMPLE

```
DefStack 25000
```

1.11 The »DefPri« command line argument

This specifies the default priority to launch executables at.

This option is ignored by ARexx events.

If not specified, this option defaults to zero.

TEMPLATE

```
DEFPRI/K/N
```

EXAMPLE

```
DefPri -1
```

1.12 The »CronPri« command line argument

This is the priority that CyberCron should run at.

If not specified, this option defaults to no change. I.e., CyberCron will remain that the priority it was started at.

ToolPri is accepted as an alias for CronPri. This is for conformity with the Style Guide.

TEMPLATE

```
TOOLPRI=CRONPRI/K/N
```

EXAMPLE

```
CronPri 4
```

1.13 The »PortName« command line argument

This option specifies the ARexx portname that CyberCron will use.

If not specified, CyberCron defaults to a portname of CYBERCRON.

CyberCron will not let you start it multiple times with the same port name. It does, however, support the use of concurrent CyberCron tasks which each have their own portname.

Using this option, one could have multiple crontab files. One might also choose to have one CyberCron task that handles their crontab file while a second is used as a server for script commands such as the at command.

TEMPLATE

```
PORTNAME/K
```

EXAMPLE

```
PortName CyberCronAtServer
```

1.14 Creating a crontab file

Lines in this file must be blank, a comment, or an event. Comments are denoted by lines that start with a "#" symbol. Events take the form of

```
min hour day month dow command <command's args, if any>
```

in their simplest form. Each of min, hour, day, month and dow (day of the week) are specifications for what range of values this command is to be run at. A * indicates any value is acceptable. Numeric digits specify specific values and can be separated by a , (seperates distinct entries) or a - (used to indicate a range of entries). For example,

```
* * * * * Date
```

would run the command date every minute of every hour of every day of every month regardless of the day of the week.

```
0 * * * * Date
```

is similar, but only runs the date command on the zeroth minute of each hour.

```
1,5-10,25 * * * * Date
```

would run the date command on minutes 1, 5, 6, 7, 8, 9, 10 and 25 of each hour.

Months range from 1 (January) to 12 (December). The day of the week ranges from 0 (Sunday) to 6 (Saturday).

CyberCron also supports a number of extensions to the traditional event format. These may occur anywhere in the event, and are:

```
:NAME <name>
< <filespec>
> <filespec>
2> <filespec>
:MAILUSER <user name>
:EXECONCE
:OBEYQUEUE <queue name>
:NOLOG
:REXX
:STACK <bytes>
:PRI <priority>
:CUSTOMSH <shell name>
:SYSSH
```

Please note that these above options are parsed with the `dos.library ReadArgs()` command. Because of this, those options which take a variable must have whitespace between the option's keyword and its assignment. This generally isn't a problem, except that most people I know are used to specifying redirection as

```
>ram:foo
```

while the proper method for CyberCron would be

```
> ram:foo
```

The other implication of using `ReadArgs()` is that quotes will be stripped out, as will extra spaces. To retain spaces, you must inclose the element they proceed or follow with quotes. For example, one might use

```
date > "Ram:Two Spaces"
```

To retain quotes, you will have to quote the argument as you would for spaces, but escape a quote character withing the quoted string. For example.

```
"This has a *" quote in it"
```

1.15 The »:NAME« event line option

Specifies the formal name the event will be known under when CyberCron is accessed via its ARexx port.

You may use the same name multiple time. However, this is not recommened.

TEMPLATE

```
:NAME/K
```

EXAMPLE

```
:Name WakeMeUp
```

1.16 The input redirection event line option

This specifies where input is to be redirected from.

TEMPLATE

</K

EXAMPLE

< "CON:0/0/300/100/Gimme some input"

1.17 The output redirection event line option

Specifies where output is to be redirected to.

If >> is used instead of >, the filespec will be appended to instead of overwritten.

TEMPLATE

>/K or >>/K

EXAMPLES

> RAM:job.out
>> RAM:job.logfile

1.18 The error redirection event line option

Specifies where standard error is to be redirected to.

If 2>> is used instead of 2>, the filespec will be appended to instead of overwritten.

Note that this is currently ignored by the event launcher per a discussion I had with Randell Jesup. Standard error support wasn't really finished for 2.0, and that makes it difficult to support this at this time.

TEMPLATE

2>/K or 2>>/K

EXAMPLES

2> RAM:job.err
2>> RAM:job.errlog

1.19 The »:MAILUSER« event line option

Specifies a user to send mail to with the output of the command being run. If the SENDMAIL command line option was not specified, this will be ignored when the event is started, and any output redirection specified will be used, instead.

The username may be any e-mail address, so mail can be sent to both local and remote systems. If your sendmail supports it, you could even specify an alias and have it send mail to multiple people.

```
TEMPLATE
:MAILUSER/K
```

```
EXAMPLE
:MailUser root
```

1.20 The »:EXECONCE« event line option

After the event has been started, remove it from the event list.

```
TEMPLATE
:EXECONCE/S
```

```
EXAMPLE
:ExecOnce
```

1.21 The »:OBEYQUEUE« event line option

Select a specific queue to schedule the event under. See the section on queues for more information.

Using this, :EXECONCE, :MAILUSER and some ARExx macros, one could simulate the at and batch commands under UNIX. (See Loren's at stuff for an example of how.)

```
TEMPLATE
:OBEYQUEUE/K
```

```
EXAMPLE
:ObeyQueue c
```

1.22 The »:NOLOG« event line option

If you specified a logfile then every time an event is started or ends a message gets sent to the log file. This turns off logging for the specific event. Handy for things one runs many, many times, in keeping the size of your logfile from exploding.

```
TEMPLATE
:NOLOG/S
```

```
EXAMPLE
:NoLog
```

1.23 The »:REXX« event line option

Start this command using ARexx rather than trying to run it as a system call. If the first character of the command is "`" then it will be started as an ARexx string rather than a command.

TEMPLATE
:REXX/S

EXAMPLE
:Rexx

1.24 The »:STACK« event line option

The stacksize to use when launching the event.

This option is ignored for ARexx events.

If the value specified is less than 2048, or this option is omitted, then the default stack size will be used instead.

TEMPLATE
:STACK/K/N

EXAMPLE
:Stack 50000

1.25 The »:PRI« event line option

The priority to launch the event at.

This option is ignored for ARexx events.

If this option is omitted then the default priority will be used.

TEMPLATE
:PRI/K/N

EXAMPLE
:PRI -5

1.26 The »:CUSTOMSH« event line option

Use a specific custom shell when launching the event. See your "Using the System Software" manual for more information on custom shells.

This option is ignored by ARexx events.

TEMPLATE

```
:CUSTOMSH/K
```

EXAMPLES

```
:CustomSH WShell
:CustomSH BootShell
```

1.27 The »:SYSSH« event line option

If no custom shell is specified, CyberCron will try and launch the command using the specified User Shell. If you do not want the User Shell to be used for a specific event, specifying this flag will cause CyberCron to use the system shell.

This option is ignored by ARexx events.

TEMPLATE

```
:SYSSH/S
```

EXAMPLE

```
:SysSH
```

1.28 CyberCron as an ARexx Function Host

CyberCron's default ARexx port name is CYBERCRON. However, this can be changed with the PORTNAME command line option.

CyberCron supports the following ARexx commands:

```
SHUTDOWN
VERSION
SUSPEND
RESUME
NEW_EVENT_FILE <filespec>
CLOSE_EVENT_FILE
NEW_LOG_FILE <filespec>
CLOSE_LOG_FILE
SHOW_STATUS
PURGE_REXX_EVENTS
ADD_EVENT <event>
DELETE_REXX_EVENT <event name>
DELETE_EVENT <event name>
LIST_EVENTS
SHOW_EVENT <event name>
SET_QUEUE_MAX <queue name> <max jobs executable at a time>
GET_QUEUE_MAX <queue name>
EVENT_NEXT_EXEC <event name>
NEXT_EVENT_TO_EXEC
EXPAND_SSSC <seconds since system creation>
SSSC_TO_ASCII <seconds since system creation>
```

1.29 The »SHUTDOWN« ARexx command

Causes CyberCron to quit. Can also be accomplished by sending a break signal to CyberCron.

Once SHUTDOWN is started, CyberCron will not accept any more ARexx commands.

It may take a while for CyberCron to actually quit, however, because it must wait for any currently running events to terminate.

QUIT is accepted as a synonym of SHUTDOWN to provide comformity with the Style Guide.

TEMPLATE
QUIT=SHUTDOWN

EXAMPLES
Shutdown
Quit

1.30 The »VERSION« ARexx command

Returns the version, revision and date of CyberCron in the form

<version>.<revision> (<day>.<month>.<year>)

This is returned in the RESULT string, so you must issue an OPTIONS RESULTS command before you can use this command.

TEMPLATE
VERSION

EXAMPLE
options results
address CYBERCRON Version
say result

1.31 The »SUSPEND« ARexx command

Stop processing events. File notifcation on the crontab file remains active, however, so changes will be processed normally.

TEMPLATE
SUSPEND

EXAMPLE
Suspend

1.32 The »RESUME« ARexx command

Resume processing of events after a SUSPEND has been issued.

TEMPLATE
RESUME

EXAMPLE
Resume

1.33 The »NEW_EVENT_FILE« ARexx command

Change the crontab file to <filespec>. This checks to make sure the file actually exists before making the change. However, parse errors will not be noticed until the current crontab has been purged and CyberCron actually starts reading the new file.

If something goes wrong (crontab file not found or its name is too long) which doesn't cause the previous crontab file to have been purged yet then this will return RC_WARN (5).

If a fatal error occurs (couldn't restart notification on the new file) and the previous crontab has been purged, but the new one not read in, this will return RC_ERROR (10).

TEMPLATE
NEW_EVENT_FILE FILESPEC/A

EXAMPLE
New_Event_File S:CronTab2

1.34 The »CLOSE_EVENT_FILE« ARexx command

Free all entries from the CronTab. Processing of entries added by the ARexx ADD_EVENT command will still continue. Use NEW_EVENT_FILE to read a crontab file again, if you so desire.

TEMPLATE
CLOSE_EVENT_FILE

EXAMPLE
Close_Event_File

1.35 The »NEW_LOG_FILE« ARexx command

Changes the logfile to <filespec>.

If an error occurs (the new name is too long, for example) then RC_WARN (5) will be returned and the old logfile, if specified, will remain active.

```
TEMPLATE
  NEW_LOG_FILE FILESPEC/A
```

```
EXAMPLE
  New_Log_File T:CronLog
```

1.36 The »CLOSE_LOG_FILE« ARexx command

Turns off logging. You can re-enable logging with the NEW_LOG_FILE command, if you so desire.

```
TEMPLATE
  CLOSE_LOG_FILE
```

```
EXAMPLE
  Close_Log_File
```

1.37 The »SHOW_STATUS« ARexx command

Returns a couple bits of information about CyberCron's status. This is in the form of

```
<activity> <crontab filespec> <logfile filespec>
```

where <activity> is either ACTIVE or SUSPENDED.

```
TEMPLATE
  SHOW_STATUS
```

```
EXAMPLE
  options results
  address CYBERCRON Show_Status
  say result
```

1.38 The »PURGE_REXX_EVENTS« ARexx command

All events added with the ARexx ADD_EVENT command will be purged. Processing of entries from the crontab file will continue normally.

```
TEMPLATE
  PURGE_REXX_EVENTS
```

```
EXAMPLE
  Purge_Rexx_Events
```

1.39 The »ADD_EVENT« ARexx command

Adds an event to CyberCron's active list. Use the same format as if you were specifying an event in a crontab file. Comments are not allowed.

If an error occurs in parsing the event, RC_ERROR (10) will be returned.

TEMPLATE

```
ADD_EVENT EVENT/A
```

EXAMPLE

```
ADD_EVENT '0 8 * * 1-5 Echo "Go to work!" > CON:0/0/200/50/Hey!/CLOSE/WAIT'
```

1.40 The »DELETE_REXX_EVENT« ARexx command

Deletes the specified event, but only if it was added with the ARexx ADD_EVENT command.

If the event was not found, RC_ERROR (10) will be returned.

TEMPLATE

```
DELETE_REXX_EVENT EVENT_NAME/A
```

EXAMPLE

```
Delete_Rexx_Event 'the event'
```

1.41 The »DELETE_EVENT« ARexx command

Same as DELETE_REXX_EVENT, but can be used to delete any event, even those read from the crontab file.

If the event was not found, RC_ERROR (10) will be returned.

TEMPLATE

```
DELETE_EVENT EVENT_NAME/A
```

EXAMPLE

```
Delete_Event 'the event'
```

1.42 The »LIST_EVENTS« ARexx command

Returns a list of all the current events CyberCron is tracking, separated by spaces. Note that the events are listed using CyberCron's internal naming scheme.

The result is returned in a result string, so you must use OPTIONS RESULTS before calling this command, or you will get an error.

If there are no events, then the string "<None>" is returned. If an error

occured, an empty string will be returned.

```
TEMPLATE
    LIST_EVENTS
```

```
EXAMPLE
    options results
    address CYBERCRON List_Events
    say result
```

1.43 The »SHOW_EVENT« ARexx command

Returns the full specification for the specified event, as it would be given in a crontab file, except that it will have the event's internal name prepended.

Note that events may not look exactly the same as when they were added. This is because CyberCron converts events into an internalized structure and unconverts them when this command is called.

If the event was not found then an empty string will be returned.

You must specify OPTIONS RESULTS before using this command, or you will get an error.

```
TEMPLATE
    SHOW_EVENT EVENT_NAME/A
```

```
EXAMPLE
    options results
    address CYBERCRON Show_Event 'the event'
    say result
```

1.44 The »SET_QUEUE_MAX« ARexx command

Set's the maximum number of jobs that may be run under the specified queue at any given time.

If an illegal queue name is specified, RC_ERROR (10) will be returned.

Setting the maximum to zero is legal, and will result in no jobs for the specified queue being executed until such a time as the queue maximum is raised to greater than zero again.

See the section on queues for more information.

```
TEMPLATE
    SET_QUEUE_MAX QUEUE_NAME/A MAX_JOBS_EXECUTABLE_AT_A_TIME/N/A
```

```
EXAMPLE
    Set_Queue_Max a 3
```

1.45 The »GET_QUEUE_MAX« ARexx command

Get the current maximum number of jobs that can be run under the specified queue at any given time.

If an illegal queue name is specified, -1 is returned.

See the section on queues for more information.

TEMPLATE

```
GET_QUEUE_MAX QUEUE_NAME/A
```

EXAMPLE

```
GET_QUEUE_MAX a
```

1.46 The »EVENT_NEXT_EXEC« ARexx command

Returns the next time at which the specified event will execute at. This is returned as the number of seconds since system creation.

If the event cannot be found, a null string will be returned.

CyberCron will look up to seven years into the future trying to find a time that satisfies the requirements of the event's timespec. This is to allow for the effect of leap years.

If CyberCron cannot determine a time at which the event would next execute, zero will be returned.

The result is returned in a result string, so you must use OPTIONS RESULTS before calling this command, or you will get an error.

TEMPLATE

```
EVENT_NEXT_EXEC EVENT_NAME/A
```

EXAMPLE

```
Event_Next_Exec 'the event'
```

1.47 The »NEXT_EVENT_TO_EXEC« ARexx command

This will scan through the event list, looking for the event that is due up for execution next. It returns a string in the form of

```
<event name> <SSSC>
```

The <SSSC> field is the seconds since system creation that the event will execute at.

If no event is found then the string "<None>" will be returned.

The result is returned in a result string, so you must use OPTIONS RESULTS before calling this command, or you will get an error.

```
TEMPLATE
  NEXT_EVENT_TO_EXEC
```

```
EXAMPLE
  options results
  address CYBERCRON Next_Event_To_Exec
  nextevent = result
  if (nextevent ~= '<None>') then
    parse var nextevent event sssc .
```

1.48 The »EXPAND_SSSC« ARexx command

This command will take the number of seconds since system creation and expand it into separate fields for seconds, minutes, etc. The result is returned in a string in the form of

```
<sec> <min> <hour> <day> <month> <year> <day of the week>
```

This is useful for scripts that want to make decisions based on the results of EVENT_NEXT_EXEC or NEXT_EVENT_TO_EXEC.

The result is returned in a result string, so you must use OPTIONS RESULTS before calling this command, or you will get an error.

```
TEMPLATE
  EXPAND_SSSC SECONDS_SINCE_SYSTEM_CREATION/A
```

```
EXAMPLE
  options results
  address CYBERCRON Next_Event_To_Exec
  nextevent = result
  if (nextevent ~= '<None>') then do
    parse var nextevent event sssc .
    Expand_SSSC sssc
    parse var result sec min hour day month year dow .
  end
```

1.49 The »SSSC_TO_ASCII« ARexx command

This command will take the number of seconds since system creation and return a human readable ASCII date string.

This is useful for scripts that want to display the results of EVENT_NEXT_EXEC or NEXT_EVENT_TO_EXEC to the user.

The result is returned in a result string, so you must use OPTIONS RESULTS before calling this command, or you will get an error.

```
TEMPLATE
  SSSC_TO_ASCII SECONDS_SINCE_SYSTEM_CREATION/A
```

EXAMPLE

```

options results
address CYBERCRON Next_Event_To_Exec
nextevent = result
if (nextevent ~= '<None>') then do
  parse var nextevent event sssc .
  Show_Event event
  eventDisp = result
  if (eventDispl ~= '') then do
    say '->' || eventDisp
    SSSC_To_ASCII sssc
    say '  next execution at' result
  end
end
end

```

1.50 How job queues work in CyberCron

CyberCron supports 26 job queues, named 'a' through 'z'. Each queue has a maximum number of jobs associated with it. If an event has a queue specified, not only must it be the appropriate time to run that event, but the number of events currently running under that queue must be less than the maximum allowed for the queue for the event to actually be executed.

Events that don't specify an :OBEYQUEUE argument are run under the default queue (not included in the 26 named queues that are user accessible), which allows an unlimited number of jobs to be executed at the same time.

All job queues are initialized with their maximum number of events to execute set to one. ARexx commands exist to read and change the maximum number for specific queues.

Jobs that could not be started because the queue maximum would be exceeded are marked as delayed. They will be run the next time the queue is available, regardless of whether or not their time spec matches the current time they are up for scheduling.

Job queues were originally added to support writing ARexx scripts that simulate the UNIX at and batch commands. However, I've tried to make the interface flexible and useful in other areas. For example, queues can be used to insure sequencing of events. Make sure that the queue you will use has its maximum set to one (which is the default for all queues). Specify when the first job is to run. Specify that the second job should run one minute after the first job. If the first job is still running when the second comes up for scheduling, the second job will be delayed. CyberCron will check every minute after that to see if the first job has completed and will run the second job when it is safe. Any number of jobs could be sequenced together using a queue in this manner.

1.51 The internal naming scheme used by CyberCron

CyberCron's LIST_EVENTS command will return event names in an internalized format. These names are in the form of 0x???????? where ? indicates a hex

digit. This is done so that events that have no formal name can be specified from ARexx. It also allows events to be specified unambiguously, which is useful if you have more than one event with the same formal name and you wish to delete/show/etc a specific one. To prevent naming conflicts, CyberCron will not let you specify a formal name that begins with the characters 0x (or 0X, as the case of the "x" doesn't matter).

When asking CyberCron to delete/show/etc an event, you can either use its internal name, or, if it has a formal name, you may use that. Internal names are cumbersome to use by hand, and are impractical to handle except by using console snap or under the control of an ARexx script. Formal names are much easier to deal with, which is why they are present. If you wanted to add a quick test event, you might give it a formal name of TestEvent and then use that to delete it instead of having to figure out its internal name and then using it to delete the event.

1.52 How to make CyberCron speak your language

When running under KickStart 2.1, CyberCron has the ability to use a translation catalog which contains the text strings it uses in a different language. These translation files are stored in the user's LOCALE:Catalogs/<language> directory. For example, the French string catalog for CyberCron would be the file:

```
LOCALE:Catalogs/français/CyberCron.catalog
```

Please note that case is important for the filename portion of the above filespec.

It is also possible to make a Catalogs sub-directory in the directory where the CyberCron executable resides. This reduces the amount of clutter in the user's LOCALE: volume. If you placed the CyberCron executable in SYS:WBStartup, then the corresponding location of the French catalog would be:

```
SYS:WBStartup/Catalogs/français/CyberCron.catalog
```

If there isn't a translation file for your language present in this distribution (and there most likely won't be as I don't know any foreign languages :->), you can make your own. To do this, you need to:

- 1) edit the dummy.ct file in the sources directory, translating it as appropriate. you should be looking at the Sources/CyberCronStrings.cd file as you do this since it contains many important notes about what the different strings are used for.
- 2) run CatComp (a Commodore developer tool which I can't include) to make the actual catalog that CyberCron will use. For example:

```
catcomp Sources/CyberCronStrings.cd <language>.ct catalog  
      locale:Catalogs/<language>/CyberCron.catalog
```

(the above is all on one CLI line, of course)

If you don't have access to CatComp then feel free to e-mail me your translation (lha and uuencode it, though, or any special accent marks, etc, will be lost) and I'll run it through CatComp. I'd like any translation files people come up with, anyway, so I can distribute them with future versions of CyberCron. Also, if you feel that one of the currently available translation files has something which could be translated in a better way, feel free to mail me those changes as well.

1.53 The history of CyberCron

Version 1.0 -- Initial release

Version 1.1 -- Maintenance release the day after 1.0 was released.
Don't remember what it fixed, though... :-)

Version 1.2

Version 1.3

Version 1.4

1.54 Changes in version 1.2 of CyberCron

It has been pointed out to me that I forgot to specify the default rexx extension of ".rexx". Thus, scripts in the form of

```
foo.rexx
```

would not be started unless you actually stated :REXX foo.rexx instead of being able to use :REXX foo alone.

Also added a semaphore lock in the GetJobNum()/FreeJobNum() routines to prevent the job table from possibly being clobbered by EndSystemJob() getting called at the same time CyberCron was trying to start another job.

1.55 Changes in version 1.3 of CyberCron

It has been requested that CyberCron become sensitive to small changes backwards in time. This is usually caused by reloading the system clock from the battery backed clock once a day (as the battery backed clock tends to keep better time on some machines). Before running the event scan, CyberCron will now check to see if time has gone backwards and will delay event scans until it reaches the next minute that it would have actually run a scan at if the time hadn't shifted backwards. There is a 5 minute threshold limit, however, so if time goes backwards more than that (for instance, daylight savings time changes state and you go back an hour) then CyberCron assumes you know what you are doing and will use the new time without delay.

I also switched the static declarations of struct RDArgs to AllocDosObject()/FreeDosObject() pairs. This should help insure that CyberCron will run on any future versions of the OS, even if it will be a few eons before I actually would have had to worry about this... :-)

1.56 Changes in version 1.4 of CyberCron

Fixed handling of commands with a space in them. Previously, they would be indicated as not found. Now they will work correctly. Also modified the log output a bit to indicate the command name portion of the command. The command is quoted with »« characters.

When CyberCron is first started, it will now spit an "I'm alive" message out to the logfile. Brian Vargyas wanted this so he could tell when his BBS had rebooted by reading through the log.

CyberCron now checks to see if it is already running and will exit gracefully if so. Added new PORTNAME option to startup arguments to allow multiple CyberCrons to be run if needed by specifying different ARexx port names for each.

Modified CCShow.rexx to accept one argument when started to be used as the portname to address. This allows one to easily display the active jobs for any CyberCron task running. If no portname is specified, CCShow defaults to CYBERCRON, which is CyberCron's PORTNAME default.

Aliased the CRONPRI startup option to TOOLPRI to try and be a bit more conformant with the Style Guide. Similarly, QUIT is now recognized as a synonym for the SHUTDOWN ARexx command.

Because CyberCron now officially supports multiple copies of itself running, I have made it use a public locking mechanism for the logfile. This way, one can have the same logfile for all instances of CyberCron and not have to worry about it getting clobbered. The job number list is also shared between concurrent CyberCrons as well.

Documentation redone as an AmigaGuide file. The hypertext nature of AmigaGuide makes it a lot easier to navigate around the CyberCron documentation and find information quickly.

Mike Sinz has provided me with wb2cli.o, a linker object that creates a CLI structure for me when CyberCron is started from the Workbench. This allows CyberCron to propagate the path that existed at LoadWB time to jobs it starts.

Updated the ErrorMessage() function to throw up an EasyRequester when CyberCron is started from the WorkBench. This was done because there is no StdErr to write errors to in this case. Once CyberCron is up and fully running, ErrorMessage() calls will not use a requester, however. At this point, only things like "Crontab changed, re-reading" are sent to ErrorMessage(). Since these are also sent to the Log() routine, the information isn't lost and it allows CyberCron to remain non-interactive.

Added localization support. Now I need to find people to translate the string catalog to other languages for me... :-)

Enhanced the `WBtoCLIargs()` routine. Specifically, it now calls `FindArg()` on each tooltype to decide if it should copy the tooltype over into the argument string. This prevents `ReadArgs()` from seeing arguments it doesn't know about and generating a "wrong number of arguments" error. This was done to make it possible to include tooltypes such as `DONOTWAIT` in CyberCron's `.info` file. This makes it easy to drop CyberCron in one's `WBStartup` drawer.

Modified the `lmkfile` to run source code through the indent filter. This helps make the code look a bit nicer by cleaning up comments as well as insuring that the proper indentation levels are in use.

By popular demand, removed the ANSI control sequences from the startup and help messages.

Noticed the `Amiga2Date()` function in `utility.library`. Changed `GetSystemTime()` to make use of it, saving 350+ bytes of code.

Found and squashed an Enforcer hit that would occur in `DoMsg()` when a routine wanted to return a null string.

Added four new `ARexx` commands. Two, `EVENT_NEXT_EXEC` and `NEXT_EVENT_TO_EXEC`, deal with determining the time at which an event will execute at. Both return their results as the number of seconds since system creation. To make dealing with this result easier, the `EXPAND_SSSC` and `SSSC_TO_ASCII` commands were added.

Updated `CCShow.rexx` to display the time each job is next scheduled to execute at by using the new `EVENT_NEXT_EXEC` and `SSSC_TO_ASCII` `Rexx` commands.

If `OwnDevUnit.library` is present, CyberCron will now try and obtain a lock on `LOG-UPDATE.LOCK` before trying to write to its logfile. This will protect its log updates from `AmigaUUCP`'s `trimfile` executable. For this to work properly, however, you must be using v2.1 or better of `OwnDevUnit.library`.

In previous versions of CyberCron, if a queue was already executing the maximum number of jobs it was set to and more jobs were due to be scheduled, then only the first job would be delayed and the rest would be skipped. Now all such jobs will be delayed.

Recompiled under `SAS/C 6.0`.

1.57 Credits

Don Nafis, author of `NazCron`, from which I pretty much stole the idea for the initial `ARexx` command set recognized by CyberCron.

Thomas Rokicki for a little bit of magic in the `GetSystemTime()` routine. (Although this is no longer used as of v1.4.)

Loren Rittle for, well, many things. He's responsible for getting me to add the stuff needed to support writing at and batch UNIX like commands (though I hope to have done it in a means general and flexible enough that

others may be able to use the event options in their own, new ways) and writing the `at.rexx` and `batch.rexx` scripts, despite the time pressure he is under right now. He also helped catch a number of bugs, offered little suggestions about functionality here and there and then there's always the `__emit()` trick in `MySPrintfSupp()` (I nearly cried when I saw it in his `ls-4.5` code --- it never dawned on me to do anything so simple :->).

Mike Sinz for providing the `wb2cli.o` linker module. This actually makes running CyberCron from the Workbench a useful thing to do.

Mike Koch for drawing the icon that I finally selected for CyberCron. I'd also like to thank the other people who sent in icons, particularly Pang Sie Piau, who sent me a couple rounds of icons as I made suggestions to him. His icons have a much better KickStart 2.0 look to them, but in the end I decided that Mike's better denotes the idea of a cron.

Amish Dave for moral support and helping me clean up the documentation a bit.

Sebastiano Vigna and Paolo Lenzi for the Italian translation catalog.

Xavier Drèze for the French translation catalog.

1.58 Contacting the author

If you happen to find a bug or have a suggestion for CyberCron, or just want to say "hey, cool program", please contact me using one of the ways listed below. Even if you wanna say "CyberCron sucks", let me know and be sure to say why you feel this way so that I might be able to fix what you think is wrong with the program.

These electronic forms are the most preferred means of contacting me. They will get you a response pretty quick.

e-mail: `caw@miroc.chi.il.us`
BIX: `caw`

Snail Mail is pretty slow and I'm not known for being very good about responding to it... :-)

Christopher A. Wichura
5450 East View Park
Chicago, Il. 60615
USA

You can also reach me by phone. However, please try to limit your calling to evening hours (I'm in the central time zone). If I'm not home and you leave a message, call back again anyway. Around here, one tends to get maybe 5% of the messages left for them, if lucky... :-)

(312)/684-2941

1.59 The use of »OPTIONS RESULTS« in an ARexx script

ARexx requires that one use the OPTIONS RESULTS command before making any calls that return a result string (as opposed to a result code/number).

1.60 What file notification does under KickStart 2.04 or greater

File notification is a new feature of KickStart 2.04. It allows a program to ask the operating system to tell it when a file has been created, changed or deleted.

By using file notification, it is no longer necessary for programs such as crons to constantly check if a critical configuration file has been updated. This results in a significant increase in system performance.

1.61 NazCron

NazCron is a cron for the Amiga that was written by Don Nafis. It was released as Shareware.

CyberCron has strong roots in NazCron, primarily in it's ARexx command set. Almost all its commands are the same as under NazCron. The new commands I added to cover features in CyberCron that NazCron does not have are done in a style very similar to the way NazCron handled things.

Please note that CyberCron is based on NazCron in spirit only. No code from NazCron has been incorporated into CyberCron.

If you are someone still using KickStart 1.3 (god forbid!) and are looking for a rather nice cron, you might consider trying to find a copy of NazCron.

The last information I have on how to contact Don Nafis is from the NazCron documentation, dated 1989. It is as follows:

Don Nafis
Nazlo Associates Ltd.
P.O. Box 1515
Laurel Springs, NJ 08021

(609) 228-8088 Voice

(609) 227-8278 BBS -
Viva! Amiga!
3/12/24 - 8/N/1 - 24 hours
130 Megabytes Online

Compuserve 70656, 133

1.62 The »at« command

The UNIX 'at' command executes a command (or a sequence of commands) at a specific time. For example, you can do something like

```
prompt>at next tuesday 8
echo "Tomorrow is your wife's birthday. Did you get the present?"
EOF (the keystroke that indicates end of file, CTRL-\ on the Amiga)
```

and at 8am next tuesday the echo command would be run. One might think that the output of this echo command would never be seen, right? Wrong. 'At' allows one to specify where the output is to be redirected. If it is not redirected then the system e-mails the output of the command to the user who submitted the 'at' job.

For a more complete description, consult the Unix man pages on the 'at' command.

1.63 The »Batch« command

The UNIX 'batch' command is actually a special case of the at command. Instead of executing a job at a specific time, the batch command submits the job for immediate execution. Job queues insure that only a few 'batch'ed jobs are actually running at one time, however.

'batch' is intended for running jobs that require a lot of CPU time.

For more information, consult the Unix man pages on the 'batch' command.

1.64 The Commodore PIPE: device

The Commodore supplied PIPE: device can be used to route the output of one program into the input stream of another. This is done in a memory efficient way which does not require large temporary files to be created.

For more information on the PIPE: device and how to mount it, see your "Using the System Software" manual.

1.65 AMIGA User Interface Style Guide

The "Amiga User Interface Style Guide" is a reference manual Commodore has written that has many useful suggestions on making Amiga applications more similar. The idea is that it makes it easy for a user to use a new application because many of the things they have learned from previous applications will still be valid.

The style guide is published by Addison-Wesley Publishing Company, Inc. It's ISBN number is 0-201-57757-7.

1.66 System Creation

System creation on the Amiga is the zeroth minute of the zeroth hour on January 1st, 1978.