What is a core file?

and

When are they useful?

Stuart R Salzer, Senior Support Specialist
InterSystems Corporation
Internal Support notes
2019–07-30 updated 2021–04–12

The information in this document is current as versions of InterSystems products released through the first date shown in the page footers. The update date covers errors in that discovered up to that date, but not changes present in new versions of InterSystems products. Nevertheless, the details for existing products are not subject to frequent change.

Table of Contents

Core file basics

Caché, Ensemble, HealthShare, and InterSystems IRIS data platform are very reliable. The vast majority of our customers never experience any kind of failure. However, under rare conditions, processes have failed, and in doing so have produced a core file (called a process dump file on Windows and OpenVMS). The core file contains a detailed copy of the state of the process at the instant of its failure, including the processes registers, and memory (including or excluding shared memory depending upon configuration details).

The core file is, in essence, an instantaneous picture of a failing process at the moment it attempts to do something very wrong. From this picture, we can extrapolate backward in time to find the initial mistake that led to the failure. As we look back in time, our picture of the process becomes fuzzier. With more detailed cores, we can look farther back in time before the picture becomes too fuzzy.

With properly collected core files and associated information, we can often solve, and otherwise extract valuable information about the failing process. With an artificially induced core file, usually all we can say (often after hours of analysis) is “I see what happened to this process, someone artificially forced a core of the process.” An artificially induced core of a misbehaving but the extant process can be useful as a secondary source of information to fill in details of an analysis gathered from information not available in the core.

InterSystems products can be configured to record full cores on any process failure. This has no impact on performance on your day-to-day operation. All you need is to keep a significant amount of disk space free for any potential, albeit unlikely, failure. InterSystems has a good record of solving problems when a full core is available. Sometimes we discover it was an obscure hardware failure that is never going to occur again.

InterSystems products can also be configured to record little or no information for process failures. While there is no performance advantage to disabling cores, you might find an operational advantage. Cores files can contain sensitive information. If you don’t want to have a policy for securing core files, you can enable core files only after repeated failures.

Out of the box, InterSystems products install with an intermediate approach. That being limited size cores. With these small cores, InterSystems can normally identify a previously solved problem, and maybe solve simple problems. We can’t solve all problems with the default limited cores.

The primary control for determining the size and type of core you will get is `DumpStyle`. This is a parameter in your `cache.cpf` or `iris.cpf` file. There are several other Operating System specific controls.

`DumpStyle` is explained here: **`http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=RCPF_Dumpstyle`**. `DumpStyle` takes an integer value from `0` to `8` that applies to every process in a Caché, Ensemble, HealthShare, or InterSystems IRIS data platform instance, and defines what kind of core (or process dump) file is saved should a process encounter a serious error. The defined values are:

Core file basics

Code	Name	Platform	Results
0	NORMAL	UNIX	Produces full core (depending upon other settings).
		OpenVMS	Produces CACCVIO-*pid*.LOG (of limited value).
		Windows	Produces *pid*.dmp (of limited value).
1	FULL	UNIX	Produces full core (depending upon other settings).
		OpenVMS	produces CACHE.DMP (possibly very large).
		Windows	Produces cachef*pid*.dmp (possibly very large).
2	DEBUG	UNIX	Prior to Caché 2014.1, produced core with shared memory omitted, now deprecated. Best to use OS specific methods to omit shared memory.
		OpenVMS	Unimplemented.
		Windows	Reserved to InterSystems.
3	INTERMEDIATE	UNIX	Unimplemented.
		OpenVMS	Unimplemented.
		Windows	Effective 2014.1, produces cachei*pid*.dmp .
4	MINIMAL	UNIX	Unimplemented.
		OpenVMS	Unimplemented.
		Windows	Effective 2014.1, produces cachem*pid*.dmp .
5	NOHANDLER	UNIX	Do not register a signal handler. Leave all decisions about core creation up to the operating system.
		OpenVMS	Unimplemented.
		Windows	Unimplemented.
6	NOCORE	UNIX	Do not generate a core file.
		OpenVMS	Unimplemented.
		Windows	Unimplemented.
7	NOFORK	UNIX	Create a core dump (with shared memory), but do so from the original failing process, not a forked copy of the failing process.
		OpenVMS	Unimplemented.
		Windows	Unimplemented.
8	NOFORKNOSHARE	UNIX	Create a core dump without shared memory, but do so from the original failing process, not a forked copy of the failing process.
		OpenVMS	Unimplemented.
		Windows	Unimplemented.

The default DumpStyle is 0 = NORMAL, except on Windows since Caché 2014.1, where it is 3 = INTERMDIATE.

So, for this control, set it as follows:

	Limited cores	Intermediate cores	Full cores
UNIX	0	0	1 (other controls apply)
OpenVMS	0	0	1
Windows	4	3	1

There are three ways to change the value of DumpStyle. They are:

① Place this section in your cache.cpf or iris.cpf file, you will need to use your Operating System's text editor for this:

```
[Debug]
dumpstyle=1
```

The number after the equals sign is the new default DumpStyle. Restart Caché, Ensemble, HealthShare, or InterSystems IRIS data platform. This is effective for all processes, and defines a new default for all processes if you don't override with method ② or ③ below.

② Issue the command:

```
SET old=$SYSTEM.Config.ModifyDumpStyle(1)
```

The number in parenthesis is the new value for DumpStyle. The old value is returned. This command is effective for all new processes created after it is run. Existing processes continue to run with their prior DumpStyle.

This command became effective with Caché 2014.1. For older versions, you can use this command:

```
VIEW $ZUTIL(40,2,165):-2:4:1
```

Where the new value for DumpStyle is the final digit.

③ Issue this command, or place it in your application:

```
VIEW $ZUTIL(40,1,48):-1:4:1
```

Where the new value for DumpStyle is the final digit. This is effective only for the process issuing the command, and overrides methods ① and ②.

An often asked question is how large my cores will be? The answer is the amount of [dirty] memory used by the process at the time of failure, plus a little more to describe that memory's layout. Unfortunately, there are no simple formulæ to compute that size accurately. The best estimate depends upon whether or not you will be including shared memory.

Start with this:

$$size = base + heap + extra + \underbrace{gmheap + routine + d \times global}_{\text{if shared memory is included}}$$

where *base* is the base amount of memory needed. Start with the size of the cache[.exe] or irisdb[.exe] image.

heap is the memory used by local variables your application creates. Estimate this by taking the difference of the system variable $STORAGE when your application starts and deep inside the most memory intense loop.

extra is for some features that require extra memory. There is no definitive list, but $SORTBEGIN() and MERGE are well known to use extra memory.

gmheap is from the [config] gmheap= section on your cache.cpf or iris.cpf file. This value appears in the configuration file in kio, so multiply by 1024. Skip this if you intend to exclude shared memory.

routine is the sum of all the values from the [config] routines= section of your cache.cpf or iris.cpf file. This value appears in the configuration file in Mio, so multiply by

1048576. Skip this if you intend to exclude shared memory.

d accounts for the need to describe memory used by *global*. This value will somewhat greater than one. This actual value will vary among different versions, platforms, and the global buffer size you choose. For all 8 kio buffers on the InterSystems IRIS data platform on AIX, the value is about 1.05.

global is the sum of all the values from the `[config] globals=` section of your `cache.cpf` or `iris.cpf` file. This value appears in the configuration file in Mio, so multiply by 1048576. Skip this if you intend to exclude shared memory.

Note: As a practical matter, On most large production deployments, *global* is large enough that it dwarfs all other factors. To save core files with shared memory in a typical large production deployment, $size = 1.25 \times global$ is a reasonable estimate.

If you are concerned about the amount of disk space needed to store a core file, consider:

- If you will be running on cloud service, and you don't want to pay to keep a large amount of disk space reserved, you may have to accept limited core files. That is, core files without shared memory.
- If you will be running on a server that you control but don't want to reserve a large amount of disk space on your expensive disk array, you can purchase a cheap USB disk. You don't need a fast or redundant disk for core files. You may want to attach a hasp staple onto the cheap USB disk with expoy and padlock it to something substantial.

Most operating systems have controls to redirect cores to a common directory and control the amount of information in cores. These also should be set, and you should consider the ramifications for doing so, especially from a data privacy perspective. The following sections cover the details for individual operating systems.

Moving cores to a common directory is very useful for capacity planning, but may also make the cores more accessible to anyone wishing to exfiltrate data from your site.

Many types of problems simply cannot be solved without including shared memory in the core. Cores that include shared memory tend to be much larger than cores that do not. Most of the difference is the size of your global and routine buffers.

If you are processing sensitive information, a core file without shared memory will only contain the sensitive information being processed by the one process that failed. A core file with shared memory will also contain all the global variables recently accessed by every process. Recently might represent minutes, or considerably longer.

AIX

Full (and modern style) cores should be enabled with `smit`:

```
System Environments
>  Change / Show Characteristics of Operating System
> >  Enable full CORE dump                                  true
> >  Use pre-430 style CORE dump                            false
```

This can also be seen from the command line with:

```
# lsattr -E -l sys0 | egrep 'fullcore|pre430core'↩
fullcore     true            Enable full CORE dump                 True
pre430core   false           Use pre-430 style CORE dump           True
```

And set with:

```
# chdev -l sys0 -a fullcore=true -a pre430core=false -P↩
```

The `-P` makes the change permanent.

◆

By default core files are written to the default directory of the process at the time of process failure. Typically that is the same directory as one of your main `CACHE.DAT` or `IRIS.DAT` file. This can be changed with smit:

```
Problem Determination
> Change/Show/Reset Core File Copying Directory
```

or from the command line with:

```
# chcore -p on -l /cores -n on -d↩
```

◆

Insure the file `/etc/security/limits`, has a section with the line:

```
default:
core = -1
```

Finally, ensure that by whatever means you set up environment variables for user processes each user has `CORE_NOSHM` defined or not defined as desired. If `CORE_NOSHM=1` is defined, core files exclude shared memory. If `CORE_NOSHM=0` or not defined at all, core files include shared memory. The easy way to do this for all users is to edit `/etc/environment` to include the line:

```
CORE_NOSHM=1
```

To assign what users have and do not have shared memory cores suppressed on an individual basis, edit one of these files based upon the user and the shell they use:

```
CORE_NOSHM=1;export CORE_NOSHM    # sh in /etc/profile or $HOME/.profile
export CORE_NOSHM=1               # ksh in /etc/.kshrc or $HOME/.kshrc
export CORE_NOSHM=1               # bash in /etc/bashrc or ~/.bashrc
setenv CORE_NOSHM 1               # csh in ~/.cshrc
```

Docker

Core file creation for an InterSystems IRIS data platform docker container is controlled by the host Linux system (with a few caveats). You must plan to send core files directly to an operating system file. That file can be inside the docker container, or to a directory mapped onto the host Linux system. The advantage to sending the core file to a directory mapped onto the host Linux system is that it will survive a complete failure of the container.

Since the core file must go to an operating system file, you must disable any advanced core capturing software on the host platform. You will want to set `/proc/sys/kernel/core_pattern` with an appropriate value for both the host and container system. You should choose a relatively simple directory that you know will exist on both the host and container (`/tmp` or `/cores` are the obvious best choices). You may also want to include variables to insure that cores from multiple docker containers don't overwrite each other. Thus `/cores/core.%p.%e` is a good choice.

Host OS	Disable	see page
RedHat Linux	You must disable the Automatic Bug Reporting Tool (ABRT).	8
SuSE Linux	Up through SuSE Linux Enterprise Server 11, SuSE did not have any advanced core capturing software. So all you need to do is set `/proc/sys/kernel/core_pattern` per instructions. However, as yet we do not provide instructions for disabling the advanced core capturing software in SuSE Linux Enterprise Server 12, so SuSE 12 and later versions are currently unsuitable hosts for docker containers.	11
Ubuntu Linux	You must disable apport.	13

When you launch the container you may want to include the option to map the directory you will be using for cores to the host operating system. Thus:

```
# docker run ··· -v /cores:/cores ··· ↩
```

If you don't include `-v /cores:/cores` any core files created by a process failure inside the docker container, will survive only as long as the docker container is running. If the mapping given by the `-v` option is not symmetrical, that is the value to the left and right of the colon are different, you may fail to capture some cores.

Set the corefile size ulimit. Since this is a runtime decision, add the following to the docker run command:

```
# docker run ··· --ulimit core=-1 ··· ↩
```

HP–UX

Enable placing cores in a common directory with extended naming with:

```
# coreadm -e global -g /cores/core.%p.%f↩
```

`%p` places the pid in the pathname, `%f` places the name of the executable (such as cache or iris) in the pathname. See:

```
% man 1m coreadm↩
```

for more options.

◆

Review if shared memory has been enabled in core files with:

```
# /usr/sbin/kctune core_addshmem_read↩
# /usr/sbin/kctune core_addshmem_write↩
```

Change with:

```
# /usr/sbin/kctune core_addshmem_read=1↩
# /usr/sbin/kctune core_addshmem_write=1↩
```

`1` means enable, `0` means disable. HP–UX divides shared memory into two types. In general InterSystems only uses write shared memory, but we recommend setting both types the same.

◆

On HP–UX the core size is limited by the `maxdsiz_64bit` kernel parameter. Make sure that it is set high enough that a full core can be generated.

Review with:

```
# /usr/sbin/kctune maxdsiz_64bit↩
```

Set with:

```
# /usr/sbin/kctune maxdsiz_64bit=4294967296↩
```

A user can further limit their core with a `ulimit -c` command. This command should be removed from `/etc/profile`, `$HOME/.profile`, and similar files for other shells unless it is your intention to limit core files.

RedHat LINUX

If you are running RHEL 6.0 or later (also CentOS), RedHat has added their Automatic Bug Reporting Tool (ABRT). As installed this is not compatible with Caché, Ensemble, HealthShare, or InterSystems IRIS data platform. You need to decide if you wish to configure ABRT to support Caché, Ensemble, HealthShare, InterSystems IRIS data platform, or disable ABRT.

Below sections labeled **ABRT** apply to use of ABRT,

while sections labeled ~~**ABRT**~~ apply to traditional use without ABRT.

◆

ABRT To make InterSystems products compatible with ABRT, determine the version of ABRT you are running:

```
# abrt-cli --version↩
```

Edit the ABRT configuration file. The name varies depending upon the version of ABRT:

ABRT 1.x: `/etc/abrt/abrt.conf`

ABRT 2.x: `/etc/abrt/abrt-action-save-package-data.conf`

If you installed Caché, Ensemble, or HealthShare with a `cinstall` command (most common), or InterSystems IRIS data platform with an `iris-install` command, find the `ProcessUnpackaged=` line, and change the value to `yes`.

```
ProcessUnpackaged = yes
```

Otherwise, if you installed Caché, Ensemble, HealthShare, or InterSystems IRIS data platform from an RPM module, find the `OpenGPGCheck=` line, and change the value to `no`.

```
OpenGPGCheck = no
```

Regardless of how you installed Caché, Ensemble, HealthShare, or InterSystems IRIS data platform, find the `BlackListedPaths=` line, and add a reference to `cstat` or `irisstat` in the *installation*`/bin` directory. If the `BlackListedPaths=` line does not exist, add it at the end with just the `cstat` or `irisstat` reference.

```
BlackListedPaths=[retain_existing_list,]installation_directory/bin/cstat
```

Save your edits, and restart `abrtd`:

```
# service abrtd restart↩
```

Configured as such, ABRT creates a new directory (under `/var/spool/abrt` or `/var/tmp/abrt`) for each process failure, and in that directory, place the core, and associated information.

When a process failure occurs, issue the command:

```
# abrt-cli --list↩        # for ABRT 1.x
# abrt-cli list↩          # for ABRT 2.x
```

This will show a list of recent process failures, and for each will give a directory specification. In each directory will be a `coredump` file, along with many other small files that collectively can be quite useful in determining the cause of the process failure.

From some other directory, enter the command:

```
% tar -cvzf wrcnumber-core.tar.gz /var/spool/abrt/directory/*↩
```

Where *wrcnumber* is the number InterSystems assigns to investigate your case. You can send us the compressed *wrcnumber*`-core.tar.gz` file.

◆

~~ABRT~~ Alternatively, you can disable ABRT with:

```
# service abrtd stop↩
# service abrt-ccpp stop↩        # ABRT 2.x only.
```

To permanently disable ABRT:

```
# chkconfig abrtd off↩
# chkconfig abrt-ccpp off↩      # ABRT 2.x only.
```

Finally you need to update `/proc/sys/kernel/core_pattern`, see the next section.

◆

~~ABRT~~ You can control where cores are deposited (unless you are using ABRT).

① If you are using ABRT, you must skip this step.
② If you have disabled ABRT, you must perform this step.
③ If you never had ABRT, this step is optional.

Edit the file `/proc/sys/kernel/core_pattern`

In the simple case, just use:

```
core
```

It is generally useful to add the pid, and name of the program generating the core with:

```
core.%p.%e
```

You might also place the cores in a common directory with:

```
/cores/core.%p.%e
```

Verify that all users have write access to directory chosen. See `man core` for more options. You should make this change permanent by creating a file in the directory `/etc/sysctl.d` with a name ending with `.conf`, and containing:

```
kernel.core_pattern=/cores/core.%p.%e
```

◆

ABRT ~~ABRT~~ You should set the `/proc/self/coredump_filter` to control the amount of memory dumped to the core. This can be in an appropriate `/etc/profile.d/`*something*`.sh` file. The command is:

```
# echo 0x33 >/proc/self/coredump_filter↩
```

The exact bitmap used depends upon the level of data you wish to collect. The meanings of the bits can be found in `man core`, samples that make sense for InterSystems products are:

Bit	Description	Need for InterSystems
0x01	Anonymous private mappings.	Always needed.
0x02	Anonymous shared mappings.	Needed for complex problems.
0x04	File-backed private mappings.	Maybe needed for problems with $ZF().
0x08	File-backed shared mappings.	Maybe needed for problems with $ZF().
0x10	Dump ELF headers.	Always needed.
0x20	Dump private huge pages.	Not currently used by InterSystems.
0x40	Dump shared huge pages.	Not currently used by InterSystems.
0x80	Dump private DAX pages (RHEL 8).	Not currently used by InterSystems.
0x100	Dump shared DAX pages (RHEL 8).	Not currently used by InterSystems.

As an alternative to placing this in a shell specific script, you can modify this during boot. These instructions only apply if you boot with `grub2`. You can test this with:

```
# grub2-install --version↩
grub2-install (GRUB) 2.02~beta2
```

Edit `/etc/default/grub`. Change the line that begins `GRUB_CMDLINE_LINUX_DEFAULT=`. If the line doesn't already exist in the file, just add it at the end. It should contain:

```
GRUB_CMDLINE_LINUX_DEFAULT="oldcmd coredump_filter=newval"
```

Note: The *oldcmd* is the old value of `GRUB_CMDLINE_LINUX_DEFAULT` (omit, if the line didn't previously exist). *newval* is the new value for `coredump_filter` in hexadecimal with a leading "`0x`".

Run:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg↩
```

◆

ABRT You should set your `ulimit -c` for all processes to unlimited. This can be set globally in the file `/etc/security/limits.conf`. Add these two lines:

```
*               soft    core            unlimited
*               hard    core            unlimited
```

SuSE LINUX

If your are running SuSE Linux Enterprise Server 12 or later, SuSE now stores all cores in the systemd journal. Core files stored in the systemd journal are transitory. They do not survive a system reboot. Cores, if needed, must be extracted from the systemd journal before any system reboot.

To list the core files currently in the systemd journal:

```
# [systemd-]coredumpctl list
```

To extract a core selected by the pid that created the core:

```
# [systemd-]coredumpctl -o core.morename dump pid↩
```

Note: the `systemd-` prefix was removed from the command name effective with SuSE 12-SP2.) It is recommended that you leave this systemd behaviour in place, and not attempt to defeat it.

If you are running an older version of SuSE Linux Enterprise (11 or earlier), you can control where cores are deposited by editing the file `/proc/sys/kernel/core_pattern`

In the simple case, just use:

```
core
```

It is generally useful to add the pid, and name of the program generating the core with:

```
core.%p.%e
```

You might also place the cores in a common directory with:

```
/cores/core.%p.%e
```

Verify that all users have write access to directory chosen. See `man core` for more options.

You can make this change permanent by appending these lines to the file `/etc/sysctl.conf`:

```
# Make this core pattern permanent (SuSE 12 breaks this, don't use):
kernel.core_pattern=/cores/core.%p.%e
```

◆

You should set the `/proc/self/coredump_filter` to control the amount of memory dumped to the core. This can be in an appropriate `/etc/profile.d/something.sh` file. The command is:

```
# echo 0x33 >/proc/self/coredump_filter↩
```

The exact bitmap used depends upon the level of data you wish to collect. The meanings of the bits can be found in `man core`, samples that make sense for InterSystems products are:

Bit	Description	Need for InterSystems
0x01	Anonymous private mappings.	Always needed.

Bit	Description	Need for InterSystems
0x02	Anonymous shared mappings.	Needed for complex problems.
0x04	File-backed private mappings.	Maybe needed for problems with $ZF().
0x08	File-backed shared mappings.	Maybe needed for problems with $ZF().
0x10	Dump ELF headers.	Always needed.
0x20	Dump private huge pages.	Not currently used by InterSystems.
0x40	Dump shared huge pages.	Not currently used by InterSystems.
0x80	Dump private DAX pages. (SuSE 15).	Not currently used by InterSystems
0x100	Dump shared DAX pages. (SuSE 15).	Not currently used by InterSystems

As an alternative to placing this in a shell specific script, you can modify this during boot. To do this use yast2. The user interface for yast2 will vary depending upon whether you are connected with a terminal interface (it will use a curses interface), or a GUI interface. These instructions try to be interface agnostic.

① After launching yast2, select System → Boot Loader from the menu.

② Select the Kernel Parameters tab.

③ Look for the Optional Kernel Command Line Parameter field.

④ If the field does not already contain coredump_filter=0x*value*, appended it to the field with a space separator. If it already contains the assignment, simply edit *value*.

⑤ Exit the menu system, and reboot.

◆

You should set your ulimit -c for all processes to unlimited. This can be set globally in the file /etc/security/limits.conf. Add these two lines:

```
*               soft    core            unlimited
*               hard    core            unlimited
```

◆

Note: It may be necessary to disable AppArmor, which blocks application behaviour which it considers unusual, and writing to a core file may be considered unusual.

```
# rcapparmor stop↵
```

Ubuntu LINUX

Ubuntu uses apport to trap all process failures, and for packages added with its installation package, create apport reports which contain encoded and compressed cores with additional information. It is possible ask apport to process unpackaged code, that is applications not installed with Ubuntu's package manager. Unfortunately, in doing so, Canonical treats the apport reports created for unpackaged code as something it can examine for the improvement of Ubuntu.

Since it is possible to extract your data from an apport report, you almost certainly do not want to enable apport processing of unpackaged code. Your only choice is to disable apport. To do this, edit `/etc/default/apport`, and edit the `enabled=` line:

```
enabled=0
```

◆

Create a file `/etc/sysctl.d/30-core-pattern.conf` (or any similar name in that directory). In that file place:

```
kernel.core_pattern=/cores/core.%p.%e
```

Insure that the directory you specify for saving cores is publicly writable, and has sufficient disk space. See `man core` for more options.

◆

You should set the `/proc/self/coredump_filter` to control the amount of memory dumped to the core. This can be in an appropriate `/etc/profile.d/`*something*`.sh` file. The command is:

```
# echo 0x33 >/proc/self/coredump_filter↩
```

The exact bitmap used depends upon the level of data you wish to collect. The meanings of the bits can be found in `man core`, samples that make sense for Caché are:

Bit	Description	Need for InterSystems
`0x01`	Anonymous private mappings.	Always needed.
`0x02`	Anonymous shared mappings.	Needed for complex problems.
`0x04`	File-backed private mappings.	Maybe needed for problems with `$ZF()`.
`0x08`	File-backed shared mappings.	Maybe needed for problems with `$ZF()`.
`0x10`	Dump ELF headers.	Always needed.
`0x20`	Dump private huge pages.	Not currently used by InterSystems.
`0x40`	Dump shared huge pages.	Not currently used by InterSystems.
`0x80`	Dump private DAX pages (16.04LTS).	Not currently used by InterSystems.
`0x100`	Dump shared DAX pages (16.04LTS).	Not currently used by InterSystems.

As an alternative to placing this in a shell specific script, you can modify this during boot. These instructions only apply if you boot with `grub2`. You can test this with:

```
# grub-install --version↩
grub-install (GRUB) 2.02-2ubuntu8.12
```

Edit /etc/default/grub. Change the line that begins GRUB_CMDLINE_LINUX_DEFAULT=. If the line doesn't already exist in the file, just add it at the end. It should contain:

```
GRUB_CMDLINE_LINUX_DEFAULT="oldcmd coredump_filter=newval"
```

Note: The *oldcmd* is the old value of GRUB_CMDLINE_LINUX_DEFAULT (omit, if the line didn't previously exist). *newval* is the new value for coredump_filter in hexadecimal with a leading "0x".

Run:

```
# grub-mkconfig -o /boot/grub2/grub.cfg↩
```

◆

You should set your ulimit -c for all processes to unlimited. This can be set globally in the file /etc/security/limits.conf. Add these two lines:

```
*               soft    core            unlimited
*               hard    core            unlimited
```

macOS (OS X, Darwin)

Mac OS X was renamed OS X, and it was later renamed macOS. All these operating systems are Apple's proprietary user interface layered upon Darwin, an operating system that Apple derived from BSD UNIX and theoretically, released to the public domain. Nevertheless, Apple releases Darwin in such a way, that as a practical matter no one will ever run just Darwin.

InterSystems products only require Darwin, but since Darwin isn't practically available, all instructions are based upon the full Apple Mac OS X, OS X, or macOS.

macOS includes CrashReporter. A tool that automatically intercepts process failures, packages the failure details as text logs, and sends the data to Apple for Analysis. CrashReporter will capture process failure details for third-party software, such as Caché, Ensemble, HealthShare, and InterSystems IRIS data platform. Which, in theory, Apple might forward to InterSystems.

InterSystems does not receive CrashReporter logs form Apple, nor have we developed the ability to analyze them. InterSystems works strictly form core files. Fortunately, CrashReporter works independently from core file creation. That is, it is possible to to process a process failure through neither, either, or both CrashReporter, and core file creation.

CrashReporter preferences can be set in System Preferences → Security & Privacy, Privacy tab. The panel name and selection of boxes varies from version to version. In Mac OS X 10.4, the panel was called just Security, and there were no relevant check boxes. In those older version the user was always presented with a dialog box on any process failure, and asked if they wanted to send the data to Apple for analysis.

Depending upon the sensitivity of the data you processes, you may want to untick all the options related to CrashReprter.

◆

The method for enabling cores in macOS has undergone significant changes from version to version. See the following chart, and use the appropriate method for your version.

Release	CodeName	InterSystems versions	Method
Public Beta	Kodiak	unsupported	**Method 1:** Edit `/hostconfig`
Mac OS X 10.0	Cheetah	unsupported	
Mac OS X 10.1	Puma	unsupported	
Mac OS X 10.2	Jaguar	unsupported	
Mac OS X 10.3	Panther	Caché (PowerPC) 5.0, 5.1	

Release	CodeName	InterSystems versions	Method
Mac OS X 10.4	Tiger	Caché (PowerPC or x86 as marked) 5.0$^{\text{PowerPC}}$, 5.1$^{\text{PowerPC}}$, 5.2*, 2007.1*, 2008.1$^{\text{x86}}$, 2008.2$^{\text{x86}}$, 2009.1$^{\text{x86}}$	**Method 2:** Edit `/etc/launchd.conf`
Mac OS X 10.5	Leopard	Caché (x86) 2008.1, 2008.2, 2009.1, 2010.1	
Mac OS X 10.6	Snow Leopard	Caché (x86–64) 2010.1, 2010.2, 2011.1, 2012.1, 2012.2	
Mac OS X 10.7	Lion	Caché (x86–64) 2011.1, 2012.1, 2012.2, 2013.1, 2014.1	
OS X 10.8	Mountain Lion	Caché (x86–64) 2012.2, 2013.1, 2014.1, 2015.1	
OS X 10.9	Mavericks	Caché (x86–64) 2013.1, 2014.1, 2015.1, 2015.2, 2016.1, 2016.2	
OS X 10.10	Yosemite	Caché (x86–64) 2014.1, 2015.1, 2015.2, 2016.1, 2016.2	**Method 3:** Not automatic.
OS X 10.11	El Capitan	Caché (x86–64) 2016.1, 2016.2, 2017.1$^{\text{DEV}}$, 2017.2$^{\text{DEV}}$, 2018.1$^{\text{DEV}}$	
macOS 10.12	Sierra	Caché (x86-64) 2017.1, 2017.2, 2018.1	
macOS 10.13	High Sierra	Caché 2018.1, IRIS 2018.1, 2019.1, 2019.2, 2019.3, 2019.4, 2020.1, 2020.2, 2020.3	
macOS 10.14	Mojave	IRIS 2019.1, 2019.2, 2019.3, 2019.4, 2020.1, 2020.2, 2020.3	
macOS 10.15	Catalina	unreleased	

Method 1: For versions OS X 10.3 (Cheetah), and prior unsupported versions: Edit the file `/hostconfig`. Find the line `COREDUMPS=`, and change the value to `-YES-`.

```
COREDUMPS=-YES-
```

◆

Method 2: For versions OS X 10.4 (Tiger) to OS X 10.9 (Mavericks), edit the file `/etc/launchd.conf`, and add the line:

```
limit core unlimited
```

And reboot.

◆

Method 3: For versions OS X 10.10 (Yosemite) and newer, `/etc/launchd.conf` is eliminated. Core file generation is now half disabled. Either users must enable cores for each process with:

```
% ulimit -c unlimited↩
```

Prior to running their application, a privileged user must run:

```
# launchctl limit core unlimited↩
```

Then logout, and login again prior to starting Caché. Apple specifically does not provide a good way to automate this, as they consider the default generation of a core file to be a potential security vulnerability.

Apple does provide a way to totally disabling core file generation. This is done by editing the file `/etc/sysctl.conf`, and adding the line:

```
kern.coredump=0
```

It can be re-enabled, by removing the line, or changing the value to `1`.

OpenVMS

By default Caché, and Ensemble will only produce CACCVIO-*pid*.LOG files for failing processes. With these only relatively simple problems can be solved. These CACCVIO-*pid*.LOG files will always be placed in the processes default directory (typically the directory of a CACHE.DAT file), and can only be redirected by changing the processes default directory.

Caché, and Ensemble may also produce CERRSAVE-*pid*.LOG files. These are similar to CACCVIO-*pid*.LOG files. Usually, you do not need to concern yourself with the difference. In some cases Caché, and Ensemble will produce both files in response to a failure. In all cases seen so far, the CACCVIO-*pid*.LOG file is produced first with the full context of the error, while the CERRSAVE-*pid*.LOG file is produced during final rundown of the process, and contains comparatively little information of value.

If extended process dumps (FULL dumps) are enabled, they too will be placed in the process default directory. However they can be redirected, by defining the logical name SYS$PROCDMP to point to a directory in which to store the process dump. This logical name can be defined at the /SYSTEM level. The file name will be CACHE.DMP or CSESSION.DMP.

OpenVMS also provides the logical name SYS$PROTECTED_PROCDMP. You should also define that logical name with both /EXECUTIVE_MODE and /SYSTEM. This applies to process failures of privileged images, and parts of Caché are privileged. The OpenVMS documentation will advise you to define the two logical names to different directories, and place higher security on directory corresponding to SYS$PROTECTED_PROCDMP. This is based upon the assumption that that the data processed by privileged images is more sensitive than that processed by non-privileged images. If both are sensitive, it is ok to point both logical names to the same directory.

◆

There is a history of defects effecting the creation of CACCVIO-*pid*.LOG and CERRSAVE-*pid*.LOG files as well as full process dumps. These are the most important changes.

Change	First version	Description
JLC1809	Caché 2015.2	Prior to this change most CERRSAVE-*pid*.LOG files were useless.
JO2422	Caché 2012.1	Prior to this change conditions that would generate a CERRSAVE-*pid*.LOG file always created the limited information file ignoring DumpStyle.
JLC1326	Caché 2011.1	Prior to this change registers were not included in CACCVIO-*pid*.LOG, and CERRSAVE-*pid*.LOG files on the Itanium platform. This seriously hampered our ability to solve all but simple problems with these files. We could still match with already solved problems.
JLC931 and JLC959	Caché 2007.2	Prior to these changes no useful information was recored in CACCVIO-*pid*.LOG, and CERRSAVE-*pid*.LOG files on the Itanium platform.
JO1968	Caché 5.2	Prior to this change conditions that would generate a CACCVIO-*pid*.LOG file always created the limited information file ignoring DumpStyle.

Solaris

You can enable placing cores in a common directory with extended naming with:

```
# coreadm -e global -g /cores/core.%p.%f -G all↩
```

① `%p` places the pid in the pathname.
② `%f` places the name of the executable (such as `cache`) in the pathname.
③ The `-G all` includes all types of memory, that is a full core. Omit this for a default core that still includes most shared memory. The following things can be stored in the core:

Code	InterSystems usage	In default
`stack`	Needed	yes
`heap`	Needed	yes
`shm`	Not used	yes
`ism`	Not used	yes
`dism`	Caché shared memory	yes
`text`	Useful for `$ZF()` failures	yes
`data`	Needed	yes
`rodata`	Not used	yes
`anon`	Needed	yes
`shanon`	Generally small	yes
`ctf`	Needed	yes
`symntab`	Useful for `$ZF()` failures	no
`shfile`	Not used	no

`all` includes all types of memory, `default` includes all but the last two. If you want significantly smaller cores (to save space at the expense of making fewer problems solvable), the most space is saved by removing `dism` shared memory. Do this with:

```
# coareadm -e Global -g /cores/core.%p.%f -G (default-dism)↩
```

See:

```
% main 1m coreadm↩
```

for more options.

◆

By default users have

```
% ulimit -c unlimited↩
```

You may use the `ulimit` (or `limit` command in csh) to disable cores, but `coreadm` is generally more flexible. So you should insure `ulimit` commands don't appear in `/etc/profile` or `$HOME/.profile`, or corresponding files for other shells.

Windows

The information to be included in a dumpfile for Windows is fully controlled by the `DumpStyle` parameter in the `cache.cpf` file (or other interface to changing `DumpStyle` defined above.

Testing

Local security setup among other problems can prevent a core from actually being written. It can be very useful to test if a core will actually be created under real-world conditions. To do that, enter the command:

```
USER>DO $ZUTIL(150,"DebugException")↩
```

To be certain, you should test this statement interactively, inside JOBs (assuming your application uses the JOB command), and even hiding inside an option of your application that your users will not accidentally select. Verify that you get a core file, and follow the sanity check in the next section to verify that it is a good core file.

Sanity Test

Core files (and process dumps) can be quite large, and they can contain sensitive information. Before transmitting a core file to InterSystems for analysis, it is best to perform a sanity test of core file on the system that generated it, or a very similar system.

Based upon your operating system, please perform the following sanity test:

<table>
<tr><th>OS</th><th colspan="2">Sanity test</th></tr>
<tr><td rowspan="2">AIX</td><td colspan="2"><pre># dbx cache core↩
(dbx) set $stack_details↩
(dbx) where↩
(dbx) quit↩</pre></td></tr>
<tr><td colspan="2">Send us the output from the above commands when opening a problem with the WRC. If you do not have dbx installed on your system, just open a new problem.</td></tr>
<tr><td rowspan="2">HP-UX</td><td><pre># gdb cache core↩
(gdb) frame 0↩
(gdb) while 1↩
 > info frame↩
 > up↩
 > end↩
(gdb) quit↩</pre></td><td><pre># adb cache core↩
adb> $c↩
adb> $q↩</pre></td></tr>
<tr><td colspan="2">Send us the output from one of the two command sets above depending upon which debugger you have available. If you have both, gdb (actually Wildebeest) is preferred.</td></tr>
<tr><td>RedHat Linux</td><td rowspan="3">Use this common sanity test for all flavours of Linux.</td><td rowspan="3"><pre># gdb cache core↩
(gdb) frame 0↩
(gdb) while 1↩
 > info frame↩
 > up↩
 > end↩
(gdb) quit↩</pre>
Send us the output from the above command when opening a problem with the WRC. If you do not have gdb installed on your system, just open a new problem.</td></tr>
<tr><td>SuSE Linux</td></tr>
<tr><td>Ubuntu Linux</td></tr>
</table>

<table>
<tr><th>OS</th><th colspan="2">Sanity test</th></tr>
<tr><td rowspan="2">macOS (Darwin)</td><td>

```
# lldb↵
(lldb) target create -c core↵
(lldb) thread backtrace all↵
(lldb) quit↵
```

</td><td>

```
# gdb cache core↵
(gdb) frame 0↵
(gdb) while 1↵
 > info frame↵
 > up↵
 > end↵
(gdb) quit↵
```

</td></tr>
<tr><td colspan="2">Send us the output from lldb (if from OS X 10.8 (Mountain Lion) or later), otherwise send the output from gdb (for Mac OS X 10.7 (Lion) or earlier).</td></tr>
<tr><td rowspan="2">OpenVMS</td><td>

```
$ ANALYZE/CRASH dumpfile.DMP↵
SDA> SHOW CALL_FRAME/ALL↵
```

If you are still running OpenVMS v7.x (or earlier), the previous command will not work, instead use:

```
SDA> SHOW CALL_FRAME↵
SDA> SHOW CALL_FRAME/NEXT↵
```

Repeat the prior command until you get an error.

```
SDA> QUIT↵
```

</td><td>

```
$ ANALYZE/PROCESS dumpfile.DMP↵
DBG> SHOW CALL/IMAGE↵
DBG> QUIT↵
```

</td></tr>
<tr><td colspan="2">Send us the output from either SDA or the debugger, but the output from SDA is preferred. If you only have a CACCVIO-pid.LOG file, check that it is not empty or almost empty.</td></tr>
<tr><td rowspan="2">Solaris</td><td>

```
# mdb cache core↵
> ::stackregs↵
> ::quit↵
```

</td><td>

```
# dbx cache core↵
(dbx) where↵
(dbx) quit↵
```

</td></tr>
<tr><td colspan="2">For almost all applications, InterSystems prefers the dbx debugger on Solaris, but for a sanity test, mdb is better. Send us the stack trace produced by mdb or dbx (mdb preferred) when you open a problem report with the WRC.</td></tr>
<tr><td>Windows</td><td colspan="2">Currently there is no recommended sanity check for Windows process dumps.</td></tr>
</table>

Attach the details of the sanity test to your WRC case, or e-mail to: **support@intersystems.com**.

Transmission

Be prepared to send us the full core along with support files that may be needed for your particular operating system. We need to know the exact version of Caché, Ensemble, HealthShare, or InterSystems IRIS data platform generated the core file. If you have relinked the software to include custom `$ZF()` functions, please send the executable. (Actually, it is more convenient, if you always send the executable.)

On most UNIX systems, it is also best to send the libraries, used by the executable. The likelihood we will need libraries for any given platform varies. Consult this table:

OS	Hardware	Need Libraries	Support Level
AIX	PowerPC	Unlikely	**A**
HP-UX	PA–RISC	Very Likely	**C**
HP-UX	Itanium	Likely	**A**
Linux (all flavours)	x86	Likely	**A**
Linux (all flavours)	x86_64	Likely	**A**
Linux (all flavours)	Itanium	Very Likely	**D**
macOS	PowerPC	Unlikely	**D**
macOS	x86	Unlikely	**C**
macOS	x86_64	Unlikely	**A**
OpenVMS	VAX	n/a	**D**
OpenVMS	ALPHA αXP	n/a	**B**
OpenVMS	Itanium	n/a	**B**
Solaris	x86_64	Very Likely	**B**
Solaris	Sparc	Unlikely	**B**
Tru64 UNIX	ALPHA αXP	Very Likely	**D**
Windows	x86	n/a	**A**
Windows	x86_64	n/a	**A**
Windows	Itanium	n/a	**D**

Explanation of Support levels

- **A** As of the posting of this document, InterSystems has the resources to diagnose core files on this platform.
- **B** Full support for this platform has recently lapsed. However, InterSystems still has the resources to diagnose core files on platform. Some diagnosed problems may not be corrected with an ad hoc build.
- **C** Legacy support. InterSystems may still have limited resources to diagnose core files on this platform, however, it may no longer be possible to to provide an ad hoc build to fix any defects found.
- **D** Nostalgia support. InterSystems does not maintain any resources to diagnose problems on these platforms. However some limited

capability survives. Cores on these platforms might be analysed. There is no chance that any defects found can be fixed.

Issue an `ldd` command to list the needed libraries:

```
# ldd install_directory/bin/image↩
       linux-vdso.so.1 =>  (0x00007fffd1320000)
       libdl.so.2 => /lib64/libdl.so.2 (0x00007f23e5002000)
       librt.so.1 => /lib64/librt.so.1 (0x00007f23e4dfa000)
       libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f23e4af0000)
       libm.so.6 => /lib64/libm.so.6 (0x00007f23e47ee000)
       libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f23e45d8000)
       libc.so.6 => /lib64/libc.so.6 (0x00007f23e4216000)
       /lib64/ld-linux-x86-64.so.2 (0x00007f23e521a000)
       libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f23e3ffa000)
```

The above contains sample output for RHEL 7. The output of all UNIX systems are similar. *install_directory* refers to the directory in which Caché, Ensemble, HealthShare, or InterSystems IRIS data platform is installed. *image* is `cache` for all products, prior to 2018, and `irisdb` for products since 2019.

If you are sending multiple files, it is best to place them in a compressed container file. In general `.ZIP` is best. `.tar.gz` is also reasonable. For OpenVMS creating a backup file with

```
$ BACKUP *.* [-]saveset.BCK/SAVE/DATA=COMPRESS↩
```

It can be helpful to include a manifest that explains the files being sent. Please prepare the manifest as a plain text file.

If you will be sending the data electronically, please do not encrypt the file, use an encrypted transmission method instead.

You can send core files to InterSystems by any of these methods:

Method	Security	Max size
Direct upload to WRC Application You must open a WRC investigation before uploading files, upon uploading a file, you will have the option of marking the problem for elevated security. Once a problem is marked for elevated security, all access to files associated to your investigation is restricted to staff actually working on the investigation. In addition to a 300 Mio size limit for attachments there is a 60 second time-limit, therefore your maximum upload is reduces if your effective bandwidth is less than about 42 Mbps.	Secure or Elevated	300 Mio and 60 second
E-mail In general e-mail should be avoided for all but simple problems that can be investigated without any customer data. Example: You just installed Caché on a new computer, and it fails upon startup wth a small core file. That file is reasonable to send via e-mail.	Unsecure	40 Mio

Method	Security	Max size
Our kite-works server You must request a link for uploading data for any given problem. These links expire in 30 day or less. This is the preferred method for uploading secure data. The absolute size limit is the amount of free space on our server. However, as this method is used by most of our customers, please advise if the files you intend to upload are greater than 4 Gio in size.	Secure	> 4 Gio
Our sftp server You must request a directory specific to the investigation. A directory will be created for the investigation. For elevated security problems we create a restricted access machine (or virtual machine) and enable an automated process to move any uploaded files to that machine. The absolute size limit is the amount of free space on our server, please advise if the files you intend to upload are greater-than 100 Gio in size.	Elevated	> 100 Gio
Your ftp/sftp server You must own and fully control any server from which you request we download data. InterSystems will not download data from any third-party server. Third-party servers are considered a security risk.	Up to you	?
SecurLink InterSystems can download files directly from any approved machine on your network through our SecurLink remote control facility. There is no absolute size limit. However if you are connected to the InterNet via a V.90 modem it would take us a week to download a 3 Gio core file.	Secure and Elevated	?
Physical media You can mail physical media to your local InterSystems office. InterSystems can read the media for and send the data to our Cambridge office where most core analysis is performed. Most offices can deal with USB disks, and ISO 9660 optical media. Our Cambridge office can deal with many tape formats. You should check with InterSystems first, before sending any media. If the media is sent via registered (not certified) mail, the data can be considered secure (possibly elevated).	Varies	?

It is important to remember that some of the files we want are binary files, while others are text. For some file transfer methods (especially between unlike operating systems), it is important to specify if the file is binary or text to prevent the file from being corrupted.

Index