AIX Version 7.1

Cluster Data Aggregation Tool User's Guide and Reference

SC23-6775-00

AIX Version 7.1

Cluster Data Aggregation Tool User's Guide and Reference

SC23-6775-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 25.

First Edition (September 2010)

This edition applies to AIX Version 6.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document v
Highlighting v
Case-sensitivity in v
ISO 9000. v

Cluster Data Aggregation Tool User Guide and Reference 1
Introduction to Cluster Data Aggregation Tool . . . 1
Cluster Data Aggregation Tool 1
Overview 1
Cluster Data Aggregation Tool (cdat) Command . 3
Periodic Collects. 15
Log Files 16
Collect Types 17
Default Collect Types 17
Collect Types for Centralized RAS Support . . . 18
Extending the Framework 20
Framework Helpers 22
Cluster Data Aggregation Tool Quick Start . . . 24

Notices 25
Trademarks 27

Index 29

About this document

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-sensitivity in

Everything in the AIX® operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type `LS`, the system responds that the command `is not found`. Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Cluster Data Aggregation Tool User Guide and Reference

To use the information effectively, you must be familiar with commands, system calls, subroutines, file formats, and special files. This topic is also available on the documentation CD that is shipped with the operating system.

To view or download the PDF version of this topic, select Cluster Data Aggregation Tool User Guide and Reference .

Downloading Adobe Reader: You need Adobe® Reader installed on your system to view or print this PDF. You can download a free copy from the Adobe website (www.adobe.com/products/acrobat/readstep.html).

Introduction to Cluster Data Aggregation Tool

Cluster Data Aggregation Tool provides a single instance to launch RAS debug and monitoring actions, and to collect problem determination data for multiple nodes.

The Cluster Data Aggregation Tool environment is made of a central master node and remote nodes. The Cluster Data Aggregation Tool is installed on and executed from the central master node. The central master node hosts the data collection repository, which is a new file system that contains all the collection data from multiple remote nodes. The remote nodes are where Cluster Data Aggregation Tool data are collected, which are AIX LPAR, VIOS, PowerHA™ pureScale and HMC. The Cluster Data Aggregation Tool is managed by the **cdat** command that is divided into several subcommands. The subcommands are **init**, **show**, **check**, **delete**, **discover-nodes**, **list-nodes**, **access**, **collect**, **list-types**, and **archive**.

- The **cdat init** subcommand initializes the Cluster Data Aggregation repository.
- The **cdat show** subcommand displays the content of the repository.
- The **cdat check** subcommand verifies the consistency of the repository.
- The **cdat delete** subcommand suppresses the specified collect from the repository.
- The **cdat discover-nodes** subcommand retrieves the names of LPAR connected to the specified list of HMC.
- The **cdat list-nodes** subcommand displays the list of known nodes that are present in the nodes.txt file.
- The **cdat access** subcommand sets up access authorization to the specified remote nodes.
- The **cdat collect** subcommand starts analysis tools on remote nodes and collects results at the end of the tool execution.
- The **cdat list-types** subcommand displays the list of installed collect types along with their descriptions.
- The **cdat archive** subcommand creates a compressed archive (tar.Z) of collects stored in the repository.

Cluster Data Aggregation Tool

AIX Cluster Data Aggregation Tool provides a single instance to launch RAS debug and monitoring actions, and to collect problem determination data for multiple nodes. The Cluster Data Aggregation Tool environment is made of a central master node and remote nodes.

Overview

AIX Cluster Data Aggregation Tool provides a single instance to launch RAS debug and monitoring actions, and to collect problem determination data for multiple nodes.

. The Cluster Data Aggregation Tool is installed on and executed from the central master node. The central master node hosts the data collection repository, which is a new file system that contains all the collection data from multiple remote nodes. The remote nodes are where Cluster Data Aggregation Tool data are collected, which are AIX LPAR, VIOS, PowerHA pureScale and HMC.

Cluster Data Aggregation Tool has following features.

- Captures problem determination data across multiple nodes.
- Supports sending data gathering tool on remote nodes.
- Provides extensible plug-in feature that supports new data collection types for user.
- Integrates with RBAC to enable non-root user to collect Cluster Data Aggregation Tool data.
- Relies on SSH for secure connectivity between nodes.
- Is controlled via CLI and SMIT panel.

The Cluster Data Aggregation Tool command is named **cdat**. It is divided into several subcommands. The subcommands are **init**, **show**, **check**, **delete**, **discover-nodes**, **list-nodes**, **access**, **collect**, **list-types**, and **archive**. The **init** subcommand creates the data infrastructure and defines the user used to run all other subcommands.

The Cluster Data Aggregation Tool provides a smit interface. It can be launched from the Problem Determination menu or using the fast path **cdat** (smit cdat).

To capture problem determination data across multiple nodes, the collect framework provides the following features:

- It coordinates invocation of Cluster Data Aggregation Tool data gathering. For second failure data capture (SFDC) it replaces the need for customers to manually start data gathering across multiple nodes, and synchronizes the start and stop of data collection instance across multiple nodes.
- Retrieving RAS and monitoring data from multiple nodes, it provides an easy way to collect them from multiple remote nodes and place onto a single central node. The RAS tools need to be pushed out to multiple nodes.
- The Cluster Data Aggregation Tool framework data can be gathered from the AIX LPAR, VIOS, PowerHA pureScale and HMC
- The central node supports AIX only. The remote nodes supports HMC (Linux®), PowerHA pureScale and AIX. The central node as a host supports AIX 61L and above releases.

The data collection repository should be large enough to contain all the data collections under a single place. The default size is 10 GB.

The Cluster Data Aggregation Tool data type collection supports data collections for some tools, like **perfpmr**, **snap**, **trace**. Cluster Data Aggregation Tool framework provides a capability for plug-in additional data collection types as needed.

The Cluster Data Aggregation Tool framework is initialized by the root user. A non-root user is created during the tool initialization, and is used for subsequent tool subcommands. Additionally, the framework allows to create a non-admin user on the remote nodes that will be used during data collection. The user is assigned AIX roles with necessary authorizations, which will ensure RAS commands (**snap**, **dump**, **trace**, **iptrace**, **perfpmr**) can be executed by this user.

The following requirements are supported for Centralized RAS PowerHA pureScale services:

- Configuration/de-configuration of the PowerHA pureScale clients. The Cluster Data Aggregation Tool master node logs into each PowerHA pureScale client and configures or deconfigures syslog, errorlog and PowerHA pureScale binding information on PowerHA pureScale clients. PowerHA pureScale binding information includes the IP address of PowerHA pureScale server, the management port and command port of the PowerHA pureScale service and transport type **INTERNET**.

- Collection of the PowerHA pureScale logs. The Cluster Data Aggregation Tool master node collects logs from the PowerHA pureScale server and automatically removes previous collections on the master node that are old enough based upon the user-specified policy.

The core of the Cluster Data Aggregation Tool framework is a standalone command line base. The root user or the specified non-admin user is able to initialize The Cluster Data Aggregation Tool and capture data by running commands.

The Cluster Data Aggregation Tool configuration is simple. The Cluster Data Aggregation Tool Framework supports configuration changes, such as a user may want to add or remove nodes or change the configuration information for a node.

Cluster Data Aggregation Tool (cdat) Command

The Cluster Data Aggregation Tool command is named **cdat**.

Purpose

The Cluster Data Aggregation Tool is divided into several subcommand.

Description

The **cdat** command is divided into several subcommand. The subcommand are **init**, **show**, **check**, **delete**, **discover-nodes**, **list-nodes**, **access**, **collect**, **list-types**, and **archive**. Only the **init** subcommand needs to be executed by the privileged user (root). The **init** subcommand creates the data infrastructure and defines the user used to run all other subcommand.

Note: To prevent concurrent accesses to the **nodes.txt** file or to the **collect.xml** file, running multiple instances of the **cdat** command on the same repository is forbidden and the repository is protected by a lock file.

Syntax

```
cdat -h <subcommand> [<options....>]
```

Flags

-h Displays command usage.

subcommand Specifies the subcommand name, among:

- init
- show
- check
- delete
- discover-nodes
- list-nodes
- access
- collect
- list-types
- archive

Output

If used with **-h** or with an unknown subcommand, the command prints the help, otherwise the output is the subcommand output.

Return Code

If used with **-h** or with an unknown subcommand, the return code is **1**. Otherwise the return code is the return code of the subcommand.

cdat init subcommand

To initialize the Cluster Data Aggregation repository the **init** subcommand is used.

Purpose

The **init** subcommand initializes the Cluster Data Aggregation repository.

Description

The **init** subcommand initializes the Cluster Data Aggregation repository. This subcommand must be run by the privileged user **root**.

You can specify the name of the directory for the repository (**/cdat** by default). You can request to create a specific file system. In that case, you can specify the name of the volume group and the size. You can specify the logical volume. It also allows you to define which user will run the **cdat** command to collect data. The default user is **cdat**.

Syntax

```
cdat init -h
cdat init [-c [-g VGName] [-s FSSize]] [-d Directory] [-l LVName] [-u User]
```

Flags

-h	Displays command usage.
-d `Directory`	Specifies the directory to use as the repository.
-u `User`	Specifies the user to use to run cdat subcommand.
-c	Creates a logical volume mounted on the cdat directory path.
-g `VGName`	Selects the volume group to use to create the logical volume.
-l `LVName`	Specifies the name of the new logical volume to use.
-s `FSSize`	Specifies the size of the logical volume.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. The command displays the name of the cdat user and requests for a password. The command displays the name of the directory used to store the collect data.

Example

```
# cdat init
Creating user "cdat"
Changing password for "cdat"
cdat's new password: *********
Re-enter cdat's new password: ********
creating directory "/cdat"
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command fails to create the user or to set his password, the return code is 2. If the command fails to create the directory, the return code is 3. If the command fails to create the logical volume, the return code is 4. Otherwise the return code is 0.

cdat show subcommand

To display the content of the Cluster Data Aggregation repository ,the **show** subcommand is used.

Purpose

The **show** subcommand displays the content of the Cluster Data Aggregation repository.

Description

The **show** subcommand displays the content of the Cluster Data Aggregation repository. A first level of verbosity only displays global collects information. A second level also displays the nodes information.

You can specify to display by node or by collect id (by default). You can specify a collect id, a PMR number, or a node to filter the output. You can use a verbose mode to display more information.

Syntax

```
cdat show -h
cdat show [-v]
cdat show [-v] Id
cdat show [-v] -p PMR
cdat show [-v] -n [Host]
```

Flags

-h	Displays command usage.
-v	Enables verbose mode. Displays node information.
-n	Displays the list ordered by node.
Id	Specifies the name of the collect.
-p	Specifies the PMR number of the collect.
PMR	

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise, the command displays information asked by user.

cdat show without parameter displays the list of collects:

```
# cdat show
Repository: /cdat
Local user: cdat

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000001/

2: 20090212-18:30:25+0200

    Gather system configuration information with snap for analysis
    PMR: 12345,589,235
    Location: /cdat/00000002/
```

With **-v** the parameter output is more verbose and displays nodes involved for each collect:

```
# cdat show -v
Repository: /cdat
Local user: cdat

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000001/

    node1:
        type : VIOS
        user : padmin
        machine id: 000069EAD300
        lpar id : 1
        timezone : CEST

    node2:
        type : LPAR
        user : root
        machine id: 000069EAD300
        lpar id : 2
        timezone: CEST

    node3:
        type : LPAR
        user : root
        machine id: 000069EAD300
        lpar id : 4
        timezone : CDT

2: 20090212-18:30:25+0200

    Gather system configuration information with snap for analysis.
    PMR: 12345,589,235
    Location: /cdat/00000002/
[...]
```

The information for only one collect can be displayed by providing the collect id:

```
# cdat show 1
Repository: /cdat
Local user: cdat

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble.
    PMR: 12345,678,901
    Location: /cdat/00000001/
```

The **-v** option is also available to display nodes information.

The list can also be displayed by node instead of by collect:

```
# cdat show -n
Repository: /cdat
Local user: cdat

node1:

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000001/
```

```
2: 20090212-18:30:25+0200

    Gather system configuration information with snap for analysis.
    PMR: 12345,589,235
    Location: /cdat/00000002/

node2:

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000001/

2: 20090127-12:52:07+0200

    Collect IP trace analysis
    PMR: 12345,678,901
    Location: /cdat/00000002/
[...]
```

The information for one given node is available using the **-n** option:

```
# cdat show -n node1
Repository: /cdat
Local user: cdat

node1:

1: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000001/

2: 20090212-18:30:25+0200

    Gather system configuration information with snap for analysis.
    PMR: 12345,589,235
    Location: /cdat/00000002/
```

The **-v** option is also available to display nodes information for each collect.

The list of collects for a given PMR is available using the **-p** option:

```
# cdat show -p 12345,678,901
Repository: /cdat
Local user: cdat

5: 20090127-12:23:45+0200

    Collect perfpmr data to identify the cause of performance trouble
    PMR: 12345,678,901
    Location: /cdat/00000005/

8: 20090212-18:30:25+0200

    Gather system configuration information with snap for analysis.
    PMR: 12345,678,901
    Location: /cdat/00000008/
```

The **-v** option is also available to display nodes information for each collect.

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command fails, the return code is 2. Otherwise the return code is 0.

cdat check subcommand

To check consistency of the Cluster Data Aggregation repository, the **check** subcommand is used.

Purpose

The **check** subcommand checks consistency of the Cluster Data Aggregation repository.

Description

The **check** subcommand checks consistency between the **cdat.xml** file that contains the description of the repository and the real content of the Cluster Data Aggregation repository.

If you specify the **-d** option, it can correct possible inconsistencies (this operation is interactive).

Syntax

```
cdat check -h
cdat check [-d]
```

Flags

-h	Displays command usage.
-d	Specifies the files that must be corrected if required.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. During the processing, the command displays the list of discovered inconsistencies. It asks for confirmation before repairing.

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command detects some inconsistencies, the return code is 2. Otherwise the return code is 0.

cdat delete subcommand

To remove the specified collects from the Cluster Data Aggregation repository , the **delete** subcommand is used.

Purpose

The **delete** subcommand removes the specified collects from the Cluster Data Aggregation repository.

Description

The **delete** subcommand removes entries from the **cdat.xml** file and from the Cluster Data Aggregation repository for the specified collects.

Each collect is identified by an id. Either you can specify a collect id to suppress the specified collect or you can specify a PMR number to suppress all the collects relative to the specified PMR number.

Syntax

```
cdat delete -h
cdat delete -p PMR
cdat delete Id
```

Flags

-h	Displays command usage.
Id	Specifies the id of the collect to delete.
-p	Specifies the PMR number of the collects to be deleted.
PMR	

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise, the commands displays the list of collects it is deleting.

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command fails to delete the collect, the return code is 2. Otherwise the return code is 0.

cdat discover-nodes subcommand

The **discover-nodes** subcommand retrieves the LPAR name of all nodes connected to one or more given HMC or IVM.

Purpose

To retrieve the LPAR name of all nodes connected to one or more given HMC or IVM , the **discover-nodes** subcommand is used.

Description

The **discover-nodes** subcommand retrieves the name of LPAR connected to the specified list of HMC. You can specify a VIOS instead of an HMC in case an IVM replaces an HMC. You can also specify an LPAR node to retrieve WPAR running on this LPAR. You can specify the file where the list of found nodes is stored. By default, the list is stored in the nodes.txt file under the directory specified with the **cdat init subcommand** (/cdat by default). You can execute the **cdat discover-nodes subcommand** several times and you can choose to append or to overwrite the file. Multiple instances of the same node are not recorded in the file.

The result of this subcommand can be used as an input to the **access** and **run** sub-commands. This subcommand retrieves LPAR names, which means if the LPAR name is not the same as the host name the user will have to edit the file to set the real host name

Syntax

```
cdat  discover-nodes -h
cdat  discover-nodes [-a|-w] [-f File] -n Type:[User@]Node ...
```

Flags

-h	Displays command usage.
-w	Specifies that the file must be overwritten.
-a	Specifies that new nodes must be appended to the file.
-f	Specifies the file where the nodes must be stored.
`File`	
-n	Specifies a list of nodes, **Type** is one of :
`Type:[User@]Node`	• HMC • VIOS • LPAR Define the HMC, VIOS, or LPAR to connect to and possibly the user used to connect to.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise, the command requests the password for the **hscroot** user (privileged user on an HMC), the **padmin** user (privileged user on an IVM) , the **root** user (privileged user on LPAR) or the specified user and displays the name of the file where the nodes list is written.

Example

```
$ cdat discover-nodes -a -n HMC:uranus -n LPAR:mylpar
hscroot@uranus's Password: ******
root@mylpar's Password: *******
Updating /cdat/nodes.txt
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command cannot connect to the HMC or IVM, the return code is 2. If the command cannot write data to the file, the return code is 3. Otherwise the return code is 0.

cdat list-nodes subcommand

The **list-nodes** subcommand displays the list of known nodes (basically, the content of the file **nodes.txt**).

Purpose

The **list-nodes** subcommand displays the list of known nodes.

Description

The **list-nodes** subcommand displays the list of known nodes (basically, the content of the file **nodes.txt**). You can specify one or more node files.

Syntax

```
cdat list-nodes -h
cdat list-nodes [-f File ...]
```

Flags

-h	Displays command usage.
-f `File`	Specifies the file containing the list of nodes. Multiple files can be specified using multiple **-f** options.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise the command lists the known remote nodes.

Example

```
$ cdat list-nodes
HMC uranus
VIOS miranda
LPAR ariel
LPAR umbriel
LPAR titania
LPAR oberon
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command is not able to list the nodes, the return code is 2. Otherwise the return code is 0.

cdat access subcommand

The access authorization to remote nodes is managed by the **access** subcommand.

Purpose

The **access** subcommand manages access authorization to remote nodes.

Description

The **access** subcommand sets up access authorization to the specified remote nodes. It creates the specified users on the remote nodes if they do not already exist and it attributes to these users all the RBAC authorizations required to perform collection of RAS data. It uses the appropriate privileged user to create the user on each node. Accordingly, **hscroot** user is used on HMC, **root** on LPAR and **padmin** on VIOS. The subcommand installs the SSH public key of the **cdat** user on the remote nodes. If the SSH daemon is not available on a remote node, it uses the exec protocol (port 512) if it is available or the telnet protocol to execute commands on the remote node. You can directly specify the list of nodes in the command line or you can specify the file containing the list of nodes. You can specify a default remote user that is used if you do not specify a user for a given node. If you specify the **-d** option, the **cdat** access subcommand removes access authorization to the specified remote nodes previously set up; it also removes the remote users on the remote nodes if they were previously created by the **cdat access** subcommand.

Syntax

```
cdat access -h
cdat access [-dF] [-u User] -n Type:[User@]Node ...
cdat access [-dF] [-u User] -f File ...
```

Flags

-h	Displays command usage.
-d	Deletes authentication credentials from remote nodes.
-F	Forces operation even if it was already done.
-u	Specifies the user to create on the remote nodes.
`User`	
-n	Specifies the list of nodes to authenticate with, where **Type** is one of:
`Type:[User@]Node`	• HMC • LPAR • VIOS **Node** is the name or the IP address of the node to connect to. **User** is the user to create on this particular node (overrides -u).
-f	Specifies the file containing the list of node to authenticate with. Multiple files can be specified using multiple **-f** options.
`File`	

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise, the command asks for the **cdat** user's password and displays the connection status.

Example

```
$ cdat access -u cdat
"cdat" user password: *********
accessing cdat@uranus
accessing cdat@miranda
accessing cdat@ariel
accessing cdat@umbriel
accessing cdat@titania
accessing cdat@oberon
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command fails to connect to a remote node, the return code is 2. If the command fails to access nodes list file, the return code is 3. Otherwise the return code is 0.

cdat collect subcommand

To start analysis tools on remote nodes and to collect results at the end , the **collect** subcommand is used.

Purpose

The **collect** subcommand starts analysis tools on remote nodes and collects results at the end.

Description

The **collect** subcommand starts analysis tools on remote nodes and collects results at the end.

Similarly to the **access** subcommand, the list of nodes can be provided either on the command line or from a file.

Several collect types can be done in one collect. This subcommand updates the **cdat.xml** file, creates the **collect.xml** file, and get remote files from nodes to put them into the local Cluster Data Aggregation repository.

Collects might be associated with a PMR number. It makes it easier to list collects that are related to the same PMR (using the **show** subcommand) or to remove all collects related to a specified PMR (using the **delete** subcommand).

Syntax

```
cdat collect [-gqv] [-i Id] [-p PMR] [-m Comment] [-u User] -t Type[,Options] ... -n Type:[User@]Node ...
cdat collect [-gqv] [-i Id] [-p PMR] [-m Comment] [-u User] -t Type[,Options] ... -f File ...
```

Flags

-h	Displays command usage.
-i `Id`	Specifies the name of the collect.
-p `PMR`	Specifies the PMR number of this collect.
-m `Comment`	Specifies a comment for this collect.
-q	Enables quiet (non-interactive) mode. No question will be asked to the user. Useful for scheduling collects from a cron job.
-v	Enables verbose mode. Displays more status information during the collect.
-g	Grows the file-system automatically if needed.
-u `User`	Specifies the user to connect to the remote nodes.
-t `Type[,Options]`	Specifies the type of collect to run. Optionally, you can specify options related to the type of collect you want.
-n `Type:[User@]Node`	Specifies the nodes to connect to and, optionally the user to use.
-f `File`	Specifies the file containing the list of nodes to connect to. Multiple files can be specified using multiple **-f** options.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help.

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the collect fails, the return code is 2. Otherwise the return code is 0.

cdat list-types subcommand

To display the list of installed collect types and their descriptions the**list-types** subcommand is used.

Purpose

The **list-types** subcommand displays the list of installed collect types and their descriptions.

Description

The **list-types** subcommand displays the list of installed collect types along with their descriptions. It searches for collect types in the **/usr/lib/cdat/types/** and the **/var/adm/ras/cdat/** directories, and in directories specified by the **CDAT_TYPE** environment variable (separated by :) . All directories containing a **manifest.xml** file are considered as a valid collect type.

Syntax

```
cdat list-types -h
cdat list-types [-v]
```

Flags

-h	Displays command usage.
-v	Enables verbose output. Displays usage of collect types.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise the command lists the installed collect types.

Example

```
$ cdat list-types -v
List of available collect types:

trace (/usr/lib/cdat/types/trace):
Records selected system events from nodes of type AIX LPAR or
VIOS.
Usage: trace [-d duration] -- [trace_options]
-d duration duration of collect in seconds (default to 30)
trace_options AIX trace(5) command options

perfpmr (/usr/lib/cdat/types/perfpmr):
Retrieves the result of the perfpmr command from nodes of type
AIX LPAR.
Usage: perfpmr [-d duration]
-d duration duration of collect in seconds (default to 600)

snap (/usr/lib/cdat/types/snap):
Gathers system configuration information from nodes of type
AIX LPAR or VIOS.
Usage: snap [snap_options]
snap_options AIX snap(5) command options (default to -a)
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command is not able to list the collect types, the return code is 2. Otherwise the return code is 0.

cdat archive subcommand

To create a compressed archive **tar.Z** of collects stored in the repository, the **archive** subcommand can be used .

Purpose

The **archive** subcommand can be used to create a compressed archive **tar.Z** of collects stored in the repository.

Description

The **archive** subcommand can be used to create a compressed archive **tar.Z** of collects stored in the repository. It is possible to archive all collects associated with a given PMR number or a collect specified by its name.

Syntax

```
cdat archive -h
cdat archive [-f File] -p PMR
cdat archive -f File Id
```

Flags

-h	Displays command usage.
Id	Specifies the identifier of the collect to be archived.
-p PMR	Specifies the PMR number of the collects to be archived.
-f FILE	Specifies the name of the archive to be created. In case where **-p** is specified, the default filename is PMR.tar.Z, PMR is the PMR number.

Output

If used with **-h** or with an invalid parameter, the subcommand prints the help. Otherwise the command creates an archive containing all the collects corresponding to the specified PMR number or collect name. The archive contains all the directories of the collects as well as a text file (**README**) describing the collects. This text file is the output of the **cdat show -v** command on the specified PMR number or collect name.

Example

```
% cdat archive -p 12345,123,123 -f archive.tar.Z
% uncompress -c archive.tar.Z | tar tf -
README
mycollect/
mycollect/logs.txt
mycollect/trace/
mycollect/trace/fleuret_ios/
mycollect/trace/fleuret_ios/logs.txt
mycollect/trace/fleuret_ios/trcfile
mycollect/trace/fleuret_ios/trcfmt
mycollect/trace/mnffdc1/
mycollect/trace/mnffdc1/logs.txt
mycollect/trace/mnffdc1/trcfile
mycollect/trace/mnffdc1/trcfmt
mycollect/trace/sohmc/
mycollect/trace/sohmc/logs.txt
mycollect/trace/sohmc/errors.txt
mycollect/trace/mnffdc2/
mycollect/trace/mnffdc2/logs.txt
mycollect/trace/mnffdc2/trcfile
mycollect/trace/mnffdc2/trcfmt
```

Return Code

If used with **-h** or with an invalid parameter, the return code is 1. If the command is not able to create the archive, the return code is 2. Otherwise the return code is 0.

Periodic Collects

It is possible to schedule periodic data collections using the **crontab(1)** command.

For example, to run the **snap** collect type every day at midnight:

```
% crontab -e cdat
0 0 * * * /usr/bin/cdat collect -q -t snap -f /cdat/nodes.txt
```

With this configuration, **cdat** creates a new directory under **/cdat** (and a new collect id) every day at midnight that will contain the snap data for each node present in **/cdat/nodes.txt**. It is possible (but not mandatory) to overwrite previous snap collects by specifying a collect id to the collect subcommand using the -i option:

```
% crontab -e cdat
0 0 * * * /usr/bin/cdat collect -q -t snap -i my_daily_snap -f /cdat/nodes.txt
```

In this case, the same directory is used for all collects. Only the last valid snap data is kept for each node present in **/cdat/nodes.txt**. Older snap data is overwritten. Removing a previously scheduled collect can be done by running the **crontab -e cdat** command and by removing the appropriate entry from the file. Scheduled collects can be managed transparently from the SMIT menus which avoids the manipulation of the **crontab** by hand.

Log Files

Log files can be used to diagnose problems encountered during a collect.

There are two types of log files, one per-collect log file that contains synchronization information between nodes and one per-node log file that contains information about collect phases and remote commands. A log file is named logs.txt and is located in the collect directory for example **/cdat/00000001/** or the node directory for example **/cdat/00000001/trace/node1** respectively for a per-collect log file or for a per-node log file. Each line in a log file is prefixed with a timestamp of the central node.

Here is an example of a per-collect log file for the trace collect on the node1 node:

```
% cat /cdat/00000001/logs.txt
2010-07-29 09:17:42: Creating "/cdat/00000001/collect.xml"
2010-07-29 09:17:42: Retrieving node information for node1
2010-07-29 09:17:44: Starting collect type "trace"
2010-07-29 09:17:44: Creating directory "/cdat/00000001/trace"
2010-07-29 09:17:44: Creating directory "/cdat/00000001/trace/node1"
2010-07-29 09:17:44: Starting "check" phase on node1 (LPAR): pid 5570774
2010-07-29 09:17:44: Waiting for children to terminate
2010-07-29 09:17:44: pid 5570774 (node1) terminated with exit status 0
2010-07-29 09:17:44: Starting "init" phase on node1 (LPAR): pid 5570776
2010-07-29 09:17:44: Waiting for children to terminate
2010-07-29 09:17:44: pid 5570776 (node1) terminated with exit status 0
2010-07-29 09:17:44: Starting "execute" phase on node1 (LPAR): pid 5570778
2010-07-29 09:17:44: Waiting for children to terminate
2010-07-29 09:17:46: pid 5570778 (node1) terminated with exit status 0
2010-07-29 09:17:46: Starting "terminate" phase on node1 (LPAR): pid 5570780
2010-07-29 09:17:46: Waiting for children to terminate
2010-07-29 09:17:47: pid 5570780 (node1) terminated with exit status 0
2010-07-29 09:17:47: Starting "grab" phase on node1 (LPAR): pid 5570782
2010-07-29 09:17:47: Waiting for children to terminate
2010-07-29 09:17:49: pid 5570782 (node1) terminated with exit status 0
2010-07-29 09:17:49: Starting "clean" phase on node1 (LPAR): pid 5570784
2010-07-29 09:17:49: Waiting for children to terminate
2010-07-29 09:17:50: pid 5570784 (node1) terminated with exit status 0
```

Here is an example of a per-node log file for the trace collect for the node1 node:

```
% cat /cdat/00000001/trace/node1/logs.txt
*** "check" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
*** "init" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
*** "execute" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
Running remote command "LANG=C /usr/sbin/trace -a -o /tmp/cdat.trc " on "node1" as user "cdat"
Return code 0
*** "terminate" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
Running remote command "LANG=C /usr/bin/trcstop" on "node1" as user "cdat"
Return code 0
*** "grab" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
Retrieving /tmp/cdat.trc from LPAR node1 using scp
Retrieving /etc/trcfmt from LPAR node1 using scp
```

```
*** "clean" phase ***
Running "/usr/lib/cdat/types/trace/trace -d1"
Running remote command "rm -f /tmp/cdat.trc" on "node1" as user "cdat"
Return code 0
```

Collect Types

The Cluster data Aggregation framework provides you with a set of collect types.

Default Collect Types

The set of default collect type are : **snap**, **perfpmr**, **trace**, **psrasgrab**, **psrasinit**, **psrasremove**.

Default Collect Types

The set of default collect type are : **snap**, **perfpmr**, **trace**, **psrasinit**, **psrasgrab**, **psrasremove**, **psrasgrab**.

- **snap** - collects snap data from an AIX LPAR or VIOS.
- **perfpmr** - collects **perfpmr** data from an AIX LPAR.
- **trace** - tracks trace data from an AIX LPAR or VIOS.
- **psrasgrab** - saves logs from a Centralized RAS repository.
- **psrasinit** - initializes PowerHA pureScale clients for Centralized RAS support.
- **psrasremove -** unconfigures PowerHA pureScale clients for Centralized RAS support.

snap collect type

The snap collect type runs the **snap** command with the specified options on an AIX LPAR or VIOS and retrieves the content of the `/tmp/ibmsupt` directory on the central repository. The snap collect type supports all the options supported by the AIX **snap** command.

Here is an example of a snap (default is `snap -a`) collect for two nodes (one for AIX LPAR and another for VIOS) :

```
% cdat collect -t snap -n LPAR:root@lpar1 -n
VIOS:padmin@vios1
```

perfpmr collect type

The **perfpmr** collect type installs the **perfpmr** command on an AIX LPAR (For example, `perf61.tar.Z`) and runs it with `perfpmr.sh 600`.

. The **perfpmr** collect type retrieves the file produced by `perfpmr.sh -o perfdata -z` on the central repository under the `perfpmr.pax.gz` directory. The perfpmr collect type supports the option **-d <duration>** that specifies the duration of the analysis in seconds (default to 600 seconds). If there is no version of the perfpmr tool matching the Operating System level of a remote node (for example, `perf61.tar.Z` for AIX 6.1) under the `/usr/lib/cdat/types/perfpmr/` directory, the perfpmr collect type fails for that node and the user must download the appropriate version of perfpmr for the node and copy it under `/usr/lib/cdat/types/perfpmr/`.

Example:

```
Phase "check" of collect type "perfpmr" failed for node lpar1:
### BEGIN REASON
/usr/lib/cdat/types/perfpmr/perf61.tar.Z not found.
Please install a version of PERFPMR suitable for AIX 6.1.3.0 under
/usr/lib/cdat/types/perfpmr/perf61.tar.Z.
You may find it at the following URL:
ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/
### END REASON
```

Example of a perfpmr collect (duration = 60 seconds) for two AIX LPAR nodes:

```
% cdat collect -t perfpmr,"-d 60" -n LPAR:lpar1 -n LPAR:root@lpar2
```

Note: You may retrieve the perfpmr package (for instance perf61.tar.Z) here ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr and choose the appropriate version (for instance choose perf61 to download perf61.tar.Z for AIX 6.1).

trace collect type

The trace collect type runs the command **trace -a** on an AIX LPAR or VIOS.

The trace collect type sleeps for the specified number of seconds and runs the **trcstop** command. It retrieves the trace file on the central repository under `trcfile`. The trace collect type supports the option **-d <duration>** that specifies the duration of the trace in seconds (defaults to 30 seconds). Options can be passed to the AIX trace (5) command that are separated from script options by the `--` symbol. The following command can be run:

```
% cdat collect -t trace,"-d 60 -- -j 492" -n LPAR:lpar1 \
     -n LPAR:root@lpar2
```

This command runs the **trace** command with option `-j 492` during 60 seconds on nodes `lpar1` and `lpar2`.

Collect Types for Centralized RAS Support

The **psrasgrab** collect type, **psrasinit** collect type, **psrasremove** collect type are collect types for Centralized RAS Support.

The following collect types are applicable to PowerHA pureScale servers and clients . A new node type named **PSCALE** is introduced for PowerHA pureScale servers. PowerHA pureScale clients do not need a special node type since they can be standard AIX LPAR. A PowerHA pureScale server being a closed appliance, it is not possible to login to a PowerHA pureScale server or to execute shell commands on it. The only collect type supported for the PowerHA pureScale server is thus **psrasgrab**.

- **psrasgrab** collect type
- **psrasinit** collect type
- **psrasremove** collect type

psrasgrab collect type

The **psrasgrab** collect type is used to harvest logs from a Centralized RAS Repository. The **CentralizedLogService** PowerHA pureScale service must be active and ready for this collect type to work. It is only applicable to nodes of type **PSCALE**.

For more information on PowerHA pureScale see here: PowerHA pureScale

To harvest syslog and errlog logs from a Centralized RAS Repository, the **psrasgrab** collect type calls the **psrasc** command on the Cluster Data Aggregation master node. To call **psrasc**, the master node first needs to establish a binding with the **CentralizedLogService** PowerHA pureScale service. It exists only for the time of the **psrasgrab** collect type. The **psrasgrab** also deletes log records from the Centralized RAS Repository by calling **psrasc** with the **-d** option.

For more information on **psrasc** command see here **psrasc**

When specifying an existing collect identifier, the **psrasgrab** collect type can delete previously collected logs based on a specified criteria. Otherwise, previously collected logs are not deleted from the Cluster Data Aggregation repository. The **psrasgrab** collect type supports the following options:

- **-p <mgmt_port>** specifies the management port of the **CentralizedLogService** PowerHA pureScale service.
- **-P <cmd_port>** specifies the command port of the **CentralizedLogService** PowerHA pureScale service.

- **-n <number>** specifies the number of log records to collect. When this parameter is not specified, all the log records are collected.
- **-t <time>** specifies the maximum amount of time (in minutes) that logs are kept in the repository. Logs older than this will be deleted when new collects are performed. If this option is not specified, logs are kept forever. Notice that this option is only applicable if **cdat collect** is called with an existing collect identifier.
- **-s <size>** specifies the maximum amount of logs (in megabytes) that are kept in the repository. Older logs will be deleted if we reach the specified size. If this option is not specified, no size limit is applied. Notice that this option is only applicable if **cdat collect** is called with an existing collect identifier.

Example to harvest logs from a Centralized AIX RAS Repository:

```
% cdat collect -t psrasgrab,"-p 65000 -P 65002" -n PSCALE:mycaserver
```

Example with an existing collect identifier:

```
% cdat collect -i mycollectid -t psrasgrab,"-p 65000 -P 65002" \
 -n PSCALE:mycaserver
```

This can be called from a cron job for instance to harvest logs every 15 minutes, removing logs that are one hour or more old and limiting the total amount of logs to 128 Megabytes:

```
% crontab -e cdat
0,15,30,45 * * * cdat collect -q -i mycollectid -t psrasgrab,"-p 65000 -P 65002 -t 60 -s
128" -n PSCALE:mycaserver
```

psrasinit collect type

The psrasinit type is used to configure PowerHA pureScale clients for Centralized RAS. It establishes PowerHA pureScale bindings with the CentralizedLogService PowerHA pureScale service and configures syslog and errlog on the clients.

The CentralizedLogService PowerHA pureScale service must be active and ready for **psrasinit** to work. The CentralizedRAS/Syslog and CentralizedRAS/Errlog log streams must have been created first on the PowerHA pureScale service, otherwise initialization of the clients will fail.

The **psrasinit** type supports the following options:

- **-s <ca_server>**: specifies the IP address or hostname of the PowerHA pureScale server hosting the CentralizedLogService PowerHA pureScale service.
- **-p <mgmt_port>**: specifies the management port of the CentralizedLogService PowerHA pureScaleservice.
- **-P <cmd_port>**: specifies the command port of the CentralizedLogService PowerHA pureScale service..

Here is an example of initialization of multiple PowerHA pureScale clients

```
% cdat collect -t psrasinit,"-s mypsserver -p 65000 -P 65002" \
 -n LPAR:root@mylpar1 -n LPAR:root@mylpar2
```

This example creates the PowerHA pureScale bindings for the **CentralizedLogService** PowerHA pureScale service hosted on PowerHA pureScale server with hostname **mypsserver** and configures syslog and errlog on AIX LPAR **mylpar1** and **mylpar2**

psrasremove collect type

The **psrasremove** type is used to unconfigure PowerHA pureScale clients previously configured with **psrasinit**. It removes bindings with the **CentralizedLogService** PowerHA pureScale service and restores previous syslog and errlog configurations.

To disestablish the PowerHA pureScale bindings on the PowerHA pureScale clients, the **psrasremove** type calls the **psbind** remove command with the local name **CentralizedLogService** on the remote nodes. To unconfigure syslog on a client, the appropriate lines from its **/etc/syslog.conf** are removed and the

syslog daemon is forced to re-read its configuration file if it is running. To unconfigure errlog, the ODM attribute **errlg_pscale_enabled** is set to **no** and the errlog daemon is restarted if it is running.

Here is an example of the removal of binding information and unconfiguration of syslog and errlog from PowerHA pureScale clients:

```
% cdat collect -t psrasremove -n LPAR:root@mylpar1 -n LPAR:root@mylpar2
```

This example deletes the PowerHA pureScale bindings with the **CentralizedLogService** PowerHA pureScale service on AIX **LPAR mylpar1** and **mylpar2** and restores previous syslog and errlog configurations.

Extending the Framework

This section describes how to extend the framework.

The user defines a new type of collect by :

1. Creating a directory with the name of the new collect type in `/var/adm/ras/cdat/` directory.
2. Creating a manifest XML file within the directory that describes the function of the newly added collect type.
3. Writing a script within the directory to perform the collect.

Format of the manifest.xml file

A manifest.xml file describes what a collect type does and what options it supports.

Here is the example of a manifest.xml file for the **trace** collect type:

```
<?xml version="1.0"?>
<manifest>
    <description>
        Retrieve trace data from remote nodes.
    </description>
    <script-arg id="d" mandatory="0" default="30"
        <description>Duration in seconds</description>
    </script-arg>
    <pass-through-arg default="-a">
        <description>AIX trace(5) command options</description>
    </pass-through-arg>
</manifest>
```

Here is the XML Schema Definition for the manifest.xml file:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="manifest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="description"/>
        <xs:element name="script-arg">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="mandatory" type="xs:integer" use="required"/>
            <xs:attribute name="default" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="pass-through-arg">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description"/>
```

```
            </xs:sequence>
            <xs:attribute name="default" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Custom Collect Scripts

Cluster Data Aggregation Tool framework defines a set of six phases for the collect operation: check, init, execute, terminate, grab, and clean. A collect type might provide a script or an executable to implement those phases. Not all phases are required, and if a given phase is not implemented, this phase is ignored. There is a synchronization point after each phase, that is the framework waits for the previous phase to finish on all nodes before starting the next phase.

- The purpose of **check** phase is to verify that the collect can be performed on the remote node. This can include checking the OS level of the remote node, checking the disk space, etc.
- The purpose of the **init** phase is to setup the environment required for the execution of the collect. This can include the installation of file sets or scripts on the remote node. It is best to use the push_file service to copy files to the remote node.
- The purpose of the **execute** phase is to start the collect process on the remote nodes.
- The purpose of the **terminate** phase is to stop the collect process on the remote nodes such that the result of the collect is available.
- The purpose of the **grab** phase is to retrieve the collected data from the remote node and to copy it into the destination directory. Use the get_file service to retrieve files or directories from a remote node as this service manages the authentication with the remote node automatically and is capable of extending the size of the repository when required.
- The purpose of the **clean** phase is to perform cleanup on the local or remote node. This can include the removal of temporary files or the removal of filesets installed during the init phase from the remote node. Notice that the clean phase is always performed, even if the collect fails or is interrupted.

A custom collect script must be provided to implement phases such as `check`, `init`, `execute`, `terminate`, `grab` and `clean`. It is not mandatory to implement all the phases. For instance, the user is not required to provide an `init` or `terminate` implementation, if he does not intend to perform any special action during those phases. The script is written in any programming language (scripts or compiled binaries). The user invokes the `push_file`, `get_file`, `mlog` and `remote_cmd` services provided by the Cluster Data Aggregation Tool framework from the custom scripts to transfer files, log messages or run commands on a remote node.

Each implemented phase of the custom script must follow the rules described for the return codes:

Return Value	Description
0	Indicates that the collect phase succeeded.
1	Indicates that the current remote node should be ignored in next phases.
more than 128	Indicates that an error occured during the execution of the phase. The framework will invoke the "clean" phase when such an error is occured.

A collect script is invoked with the following environment variables (set by the framework):

Variable Name	Description
CDAT_DEST_DIR	Specifies the directory on the master node where the data collected on the remote node should be stored.
CDAT_HOST	Specifies the host name of the remote node.
CDAT_PHASE	Specifies the phase to be executed.
CDAT_PMR	If set, specifies the PMR number associated with the current collect.
CDAT_SRVC_DIR	Specifies the path on the master node to the **get_file**, **push_file**, **remote_cmd** and **mlog** services.
CDAT_TYPE	Specifies the node type (i.e. LPAR, HMC, VIOS, or PSCALE).
CDAT_TYPE_DIR	Specifies the directory containing the script for the current collect type.
CDAT_USER	Specifies the user name to log into the remote node.

A collect script is invoked with the type options passed to the **cdat** collect command (see the **-t** flag of the cdat collect subcommand).

An example of new collect type definition that retrieves the content of the `/var/adm/ras/errlog` file from remote nodes (of type AIX LPAR) is as shown:

1. Create a new directory under `/var/adm/ras/cdat/`

   ```
   % mkdir -p /var/adm/ras/cdat/myerrlog
   ```

2. Create the `manifest.xml` file:

   ```
   % vi /var/adm/ras/cdat/myerrlog/manifest.xml
    <?xml version="1.0"?>
    <manifest>
    <description>Retrieve the content of the /var/adm/ras/errlog file.</description>
    </manifest>
   ```

   Note: You can also create localized manifest.xml files by adding local suffix (for instance manifest.fr_FR.xml file for french).

3. Create the script that fetches the `errlog` file. In the example below, only the grab phase is implemented because there is no command to execute in order to generate the file (no execute phase).

   ```
   % vi /var/adm/ras/cdat/myerrlog/myerrlog
   #!/bin/sh
   if [ $CDAT_PHASE = "grab" ]; then
       $CDAT_SRVC_DIR/get_file /var/adm/ras/errlog
       if [ $? -ne 0 ] ; then
           $CDAT_SRVC_DIR/mlog 0 "Could not retrieve errlog from $CDAT_HOST"
           exit 128
       fi
   fi
   exit 0
   ```

4. Verify if the new collect type is added and detected, by running:

   ```
   % cdat list-types
   ```

   This lists all the available collect types:

   ```
   ... myerrlog (/var/adm/ras/cdat/myerrlog)
   ```

   Retrieves the content of the `/var/adm/ras/errlog` file

5. Run the new collect type:

   ```
   % cdat collect -t myerrlog -n LPAR:root@mylpar1 -n LPAR:root@mylpar2
   ```

Framework Helpers

The framework provides a set of services to collect scripts that should be used to log messages, to execute commands on remote nodes, or to transfer files between remote nodes and the central master node.

To be portable, collect scripts should use these services to accomplish those actions instead of hand-rolling their own. These services allow the collect scripts to ignore the underlying transport protocol, which is used to connect to remote nodes (ssh, rexec, telnet, and ftp).

remote_cmd service

The **remote_cmd** service allows the execution of commands on the remote nodes.

Information about how to connect to the remote node is retrieved from the **CDAT_USER, CDAT_HOST** , and **CDAT_TYPE** environment variables. The **remote_cmd** service may use ssh, rexec, or telnet to execute the command on the remote node. If logging to the remote node requires a password (for example, no pre-authentication phase was performed for this node), the **remote_cmd** service will fail.

Syntax

```
remote_cmd <command>
```

push_file service

The **push_file** service allows to copy files or directories from the central master node to a remote node.

Information about how to connect to the remote node is retrieved from the **CDAT_USER, CDAT_HOST**, and **CDAT_TYPE** environment variables. The **push_file** service may use scp or ftp to copy the files to the remote node. If copying a file to the remote node requires a password (for example, no pre-authentication phase was performed for this node), the **push_file** service will fail.

Syntax

```
push_file <local file>....<remote directory>
```

get_file service

The **get_file** service allows to copy files or directories from the remote node to the central master node. Information about how to connect to the remote node is retrieved from the **CDAT_USER, CDAT_HOST**, and **CDAT_TYPE** environment variables.

The **CDAT_DEST_DIR** environment variable specifies the location where files must be copied. The **get_file** service may use scp or ftp to copy the files from the remote node. If copying a file from the remote node requires a password (for example, no pre-authentication phase was performed for this node), the **get_file** service will fail.

Syntax

```
get_file <remote file>....<local file>
```

A directory can be specified instead of **local file**, in this case, it is possible to retrieve several files or directories with a single call.

```
get_file <remote file>....<local directory>
```

Before retrieving the file, the **get_file** service checks if there is enough free space in the destination file system to store that file. If this is not the case, the **get_file** service extends the size of the file system automatically if the **cdat collect** was called with option **-g** option.

mlog service

The **mlog** service allows to log the diagnostic messages. These diagnostic messages are stored in the per-node log files.

Syntax

```
mlog <level> <message to log>
```

A severity level can be specified with **<level>** (integer). If the level is -1, the message is also printed on the **stderr** output (and in the **collect log** file), when the current collect phase terminates.

Cluster Data Aggregation Tool Quick Start

Before using Cluster Data Aggregation Tool you must initialize it on the master node, the central node that will gathers problem determination data.

To initialize Cluster Data Aggregation Tool, complete the following steps:

1. Log in to the Cluster Data Aggregation Tool central master node as root and run the following command **smit cdat**.
2. Select **Create the Repository**.
3. Select **Discover Nodes** to create a file that lists all of the remote nodes where you need to collect problem determination data. The default name for this file is **/cdat/nodes.txt**.
4. Select **Manage Remote Nodes** .
5. Select **Initialize Access to Remote Nodes**.
6. Specify the file name that lists all the remote nodes in the **Node filename** field.
7. Press **Enter** to perform the initialization.
8. Answer all questions when prompted and specify a password for the **cdat** user that is created on each remote node. You also have to provide the **root** password of each remote node.

After the Cluster Data Aggregation Tool master node has been initialized, you are able to collect problem determination data on the remote nodes by using default collect types such as **snap**, **trace**, **perfpmr** or **psrasgrab**.

To collect data on remote nodes, complete the following steps:

1. Log in to the Cluster Data Aggregation Tool master node as root or as cdat user and run the following command **smit cdat**.
2. Select **Collect Data from Remote Nodes**.
3. If there is a file that lists all of the remote nodes where you need to collect problem determination data, use it to specify the remote nodes. The default value for the **Node filename** field is **/cdat/nodes.txt**. If you do not have a file that lists the remote nodes, manually enter the remote nodes in the **Remote nodes** field.
4. Select the collect type you want to perform in the **Collect type** field.
5. Possibly specify options in the **Parameters** field.
6. Press **Enter** to perform the collect.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this

one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 903
11501 Burnet Road
Austin, TX 78758-3400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

/cdat/nodes.txt 15, 24
/usr/lib/cdat/types/ 13

A

access 1, 2, 3, 9, 11
archive 1, 2, 3, 14

C

cdat 3, 4, 11, 15, 16, 24
CDAT_DEST_DIR 23
CDAT_HOST 23
CDAT_TYPE 23
CDAT_USER 23
cdat.xml 8
Centralized RAS 17
check 1, 2, 8, 20
clean 2, 20
collect 1, 2, 3, 12, 23
collect log 23
collect.xml 12

D

Default Collect Types 17
delete 1, 2, 3, 8
discover-node 2, 3
discover-nodes 9
dump 2

E

execute 2, 20
Extending Framework 20

G

get_file 20, 23
grab 2, 20

H

Helpers 23
HMC 1, 2, 9, 11
hscroot 9

I

init 1, 2, 3, 4, 9, 20
iptrace 2
IVM 9

L

list-nodes 1, 2, 3, 10
list-types 1, 2, 3, 13
log file 16
LPAR 1, 2, 9, 17, 18
lpar1 18
lpar2 18

M

manifest.xml 13, 20
master node 1
mlog 20, 23

N

Node 11
nodes.txt 9, 10

P

padmin 9, 11
perfpmr 2, 17, 24
perfpmr package 17
PMR number 8
PowerHA pureScale 1, 2, 17, 18, 19
psrasgrab 17, 18, 24
psrasinit 17, 19
psrasremove 17, 19
push_file 20, 23

Q

Quickstart 24

R

RAS Support 18
remote nodes 12
remote_cmd 20, 23
root 4, 24
run 9

S

show 1, 2, 3, 5, 14
SMIT 2
smit cdat 24
snap 2, 15, 17, 24
stderr 23

T

terminate 2, 20
trace 2, 17, 18, 20, 24
trcfile 18

U

User 11

V

var/adm/ras/cdat/ 13
VIOS 1, 2, 9, 17, 18

Printed in USA

SC23-6775-00