Rocket

Rocket M204 SirTune

Reference Manual

September 2013
TUN-0704-RM-01

Notices

Edition

Publication date: September 2013

Book number: TUN-0704-RM-01

Product version: Rocket M204 SirTune

Copyright

Trademarks

Examples

This information might contain examples of data and reports. 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.

License agreement

Note

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 Fourth Avenue
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100

Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Global Technical Support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone:

North America	+1 800 755 4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

Summary of Changes

This section describes significant changes to the documentation. In most cases these changes correspond to enhancements made to the underlying product.

SirTune Version 7.2

- The SIRTUNER reporting module is no longer required and will no longer work with *SirTune* Version 7.2. Reports are now requested via a User Language program named SIRTUNEREPORT which is distributed in the SIRIUS file as part of *UL/SPF*. See "Generating Reports" on page 31.

- The report configuration parameters DATASET, MAPCORE, MPVIRT, and UPPER ("Configuration parameters" on page 38) are no longer supported.

- These report configuration parameters are added:
  - CHARACTERSPERLINE ("CHARACTERSPERLINE or CPL" on page 39)
  - LINESPERPAGE ("LINESPERPAGE or LPP" on page 42)
  - TABLEOFCONTENTS ("TABLEOFCONTENTS or TOC" on page 52)

- The effect of the TWOPASS configuration parameter is unchanged, but its name is now misleading, because two passes are no longer needed to place the table of contents at the start of the report. See "TWOpass" on page 52.

- Support for XML input to SIRTUNEREPORT is added. See "Using XML input for report configuration" on page 53.

SirTune Version 7.0

- *SirTune* now issues an internal `VIEW SYSTEM CWAIT` command and saves the output to the sample dataset. This data can then be used to produce the SYSPARM report ("REPORT SYSPARM" on page 72) by the *SirTune* Report Writer.

- As of *Sirius Mods* version 7.0, *SirTune* data collector can distinguish system methods from each other. Before this release, all system methods would show up as $CLASS_METHOD in the QUAD reports.

  Because this change for methods required a change to the sample dataset format, only the *SirTune* Report Writer version 1.6 and later can process a *SirTune* 7.0 sample dataset.

SirTune Version 1.6/6.9

The following changes correspond to changes in *SirTune* since version 1.5:

- As of *Sirius Mods* version 6.9, *SirTune* data collector functionality is integrated into the *Sirius Mods* product collection. You no longer install *SirTune* as a separate product. The *SirTune* Report Generator (SIRTUNER), however, is still installed separately. See item "3." on page 116.

  For *SirTune* users that are upgrading to this version, these are the main effects and changes to the running of *SirTune* that accommodate the new product distribution:

  - Under previous releases of *SirTune*, the SIRTUNE module was invoked instead of the *Model 204* load module during initialization. Under z/OS or OS/390, you did this as follows:

    ```
    //ONLINE  EXEC PGM=SIRTUNE,...
    ```

    Now, there is no SIRTUNE module to invoke, and the EXEC statement in your JCL should directly invoke the *Model 204* load module:

    ```
    //ONLINE  EXEC PGM=ONLINE,...
    ```

  - Once *Sirius Mods* is link edited with the *Model 204* ONLINE module, the *SirTune* data collector is initialized by default, and a new EXEC JCL statement parameter (`SIRTUNE`) is available to allow an override of the default. See "JCL for SirTune" on page 3.

  - You can use the new version of *SirTune* without modifying any of your existing *SirTune* configuration information. SIRTUNEI configuration statements and the SIRTUNED dataset may be used as-is with *Sirius Mods* version 6.9.

    Note: The SIRTUNEO dataset is obsolete under *Sirius Mods* version 6.9 and later. It is ignored if it is allocated.

- The SirTune 1.6 report writer now has a new report: SYSPARM ("REPORT SYSPARM" on page 72).

SirTune Version 1.5

The following changes correspond to changes in *SirTune* since version 1.4:

- Maintenance and support:

  - Multiple authorization criteria are supported, allowing a single copy of the SIRTUNE load module to support a set of CPUs, each with its own expiration date.

- All *SirTune* authorization zaps now contain a checksum that is stored with the zap. If you apply an authorization zap with incorrect content (for example, by misreading a fax), a *SirTune* "invalid checksum" error message is issued, and data collection is not performed (although *Model 204* will still be invoked).

  This checksum is in addition to a checksum that is displayed in a comment in the zap, which will be used with a future enhancement to *SirZap* to alert you to zap errors at the time of application.

- The SIRTUNEO output of load module SIRTUNE will show the following new information:

  - The date and time of running SIRTUNE
  - The CPU ID
  - The customer site ID
  - One or more lines displaying patch usage
  - The date and time that the authorization zap was generated at Sirius Software
  - If *SirTune* is authorized on the CPU, the expiration date of that authorization

  This information may help in various problem diagnosis; in particular, it may help you to make sure that the correct zap has been applied.

- *Model 204* V6R1 support.
- New features:
  - The *SirTune* data collector can now execute in 31-bit storage.
  - Support is added for an ONLINE module using RMODE(SPLIT) linkage.
  - The data collector has two new record types: one for end of compilation, and one, for APSY compilations, signaling switches from precompile mode to non-precompile mode. These record types help to simplify reporting, but they do not affect the actual reports.

    Note: Although this data collector change does not affect the size of the dataset, it does alter the format of the sample dataset: earlier releases of SIRTUNER cannot process a SIRTUNE 1.5 sample dataset.

  - New SIRTUNER reports: WHATC ("REPORT WHATC | WHATCM | WHATCS TOTAL | CHUNK ch_size" on page 72) and QUADC ("REPORT QUADC | QUADCM | QUADCS TOTAL | CHUNK ch_size" on page 64).

SirTune Version 1.4

The following changes correspond to changes in *SirTune* since version 1.3:

- Support for M204 V4R1, then for later *Model 204* versions as they were released (V4R2, V4R2.1, V5R1, V5R2, V5R3).

SirTune Version 1.3

The base version of this manual was converted for version 1.3.

CHAPTER 1 Introduction

The primary function of *SirTune* is to provide information to User Language programmers to make it easy for them to improve the performance of their programs. This is accomplished in two stages:

1. Data is collected in a running **ONLINE** or **BATCH204** region. This data is obtained by "polling." That is, at regular intervals *SirTune* examines the state of the **ONLINE** or **BATCH204** region and records data pertinent to this state.

   The part of *SirTune* that collects this data is called the ***data collection*** portion, which is integrated with the *Sirius Mods* for *SirTune* versions after 1.5, or which consists of a separately distributed load module named **SIRTUNE** for *SirTune* 1.5 and earlier versions.

   Occasionally in this document, the product name "*SirTune*" is also used to refer to just the data collection portion of the product.

2. After this data is collected, it is summarized and presented in a set of reports that indicate to a User Language programmer the potential performance problem areas. These problem areas can generally be refined to an arbitrary level of detail (for example, individual lines of code in a procedure).

   The part of *SirTune* that summarizes and produces reports is called the ***reporting*** portion, which consists of a User Language program named SIRTUNEREPORT, distributed in the SIRIUS file as part of *UL/SPF*. Pre-7.2 releases required a separate reporting load module distributed under the name **SIRTUNER**.

1.1 Versions

This document, the ***SirTune Reference Manual***, assumes that a site is running *SirTune* version 1.4 or later. Any documented feature or facility that requires a later version of *SirTune* will be clearly marked to indicate this.

For example, a statement that is only available in versions 1.5 and later of *SirTune* will have a sentence in its documentation such as, "This statement is only available in version 1.5 or later of *SirTune*." If a feature or parameter is not indicated as requiring any specific version of the *SirTune*, it can be assumed that it is available in all versions covered by this document: that is, versions 1.4 and later of *SirTune*.

1.2 System requirements

SirTune requires the following components to run:

- Mainframe operating systems — one of the following:
  - MVS/SP Version 1.3 or later (including MVS/XA, MVS/ESA, OS/390, and z/OS)
  - CMS (releases currently supported by IBM) under VM/ESA or z/VM
  - Hitachi VOS3
  - Fujitsu OSIV
- *Model 204* V5R1 or later, running in its **ONLINE**, **BATCH204**, or **IFAM4** configuration.
- For version 1.5 or earlier of *SirTune*, installing the *Sirius Mods* is **not** required for *SirTune*; however, if the *Sirius Mods* is installed with the *Model 204* load module, the *Sirius Mods* must be:
  - Version 6.2 or later

  For versions of *SirTune* after 1.5, the *SirTune* data collector is integrated with the *Sirius Mods*, and you **must** install the *Sirius Mods*, which must be:
  - Version 6.9 or later
- Version 7.2 or later For versions of *SirTune* after 7.0, the *UL/SPF* file SIRIUS must be downloaded to make the SIRTUNEREPORT program available for reporting.

CHAPTER 2 Collecting Data Under MVS: SIRTUNE

As of the integration of the *SirTune* data collector with the *Sirius Mods* (in *Sirius Mods* version 6.9), how you invoke *SirTune* depends on its version.

2.1 Versions of SirTune after 1.5

The data collection portion of *SirTune* is part of the *Sirius Mods* object. The data collector becomes available once the *Sirius Mods* is link edited with the *Model 204* ONLINE module.

2.1.1 JCL for SirTune

To invoke *SirTune*, The EXEC statement in your JCL should directly invoke the *Model 204* load module:

```
//ONLINE  EXEC PGM=ONLINE,...
```

This statement differs from that required for version 1.5 and earlier of *SirTune* (which is described in “Invoking the SIRTUNE module” on page 5).

In addition, the JCL that invokes *Model 204* must include a DD statement for the SIRTUNED dataset (see “The SirTune DD statements” on page 4). Then, if *SirTune* is authorized for use at your site, the *SirTune* data collector will be initialized,

If you are upgrading from an earlier version of *SirTune*, no changes are necessary to the *SirTune* DD statements you were using. However, if you specified SIRTUNEI configuration statements for the data collector, the PGM statement is ignored, since *SirTune* no longer loads the *Model 204* ONLINE or BATCH204 load module.

If you want to prevent the *SirTune* data collector from being initialized, do either of the following:

- Under z/OS or OS/390, include no SIRTUNED DD statement during initialization.
- Set the `SIRTUNE` parameter to 0.

  SIRTUNE, which is only allowed as a parameter in the EXEC JCL statement, controls whether the *SirTune* data collector is initialized at the start of a *Model 204* run.

The SIRTUNE parameter can be set to either of these values:

0 Disables initialization of the integrated *SirTune* product for a particular run.
1 Enables initialization (this is the default).

For example:

```
//ONLINE  EXEC PGM=ONLINE,PARM='SIRTUNE=Ø'
```

2.1.2 The SirTune DD statements

SirTune uses one or both of the DD statements described below.

Note: The SIRTUNEO dataset used in earlier versions of *SirTune* is obsolete in versions of *SirTune* after 1.5.

- SIRTUNED

  This required DD specifies the data set that receives the data collected by *SirTune*. This data set must have variable blocked (VB) format, and it should generally have a large block size (>10000). If DCB information is not explicitly specified, the defaults selected by *SirTune* should be adequate for all but the most extreme cases.

  If SIRTUNED is pre-allocated under ISPF or its equivalent, the recommended blocksizes are 23,476 on a 3380 and 27,998 on a 3390. If this dataset fills up, *SirTune* will simply stop collecting data for the duration of the run. A 20 megabyte SIRTUNED should be sufficient for most shops, while a 50 megabyte SIRTUNED should be sufficient for almost any requirements.

  Since the only cost of running out of space in SIRTUNED is the loss of some data, it's not worth spending a lot of time trying to size SIRTUNED exactly. Simply allocate SIRTUNED at 20 megabytes (or less if disk space is tight), and adjust the size based on experience. For more information on sizing SIRTUNED, see "Estimating SIRTUNED Size Requirements" on page 97. To keep available data from several runs, make SIRTUNED part of a GDG.

- SIRTUNEI

  This optional DD specifies a data set that contains statements that alter the *SirTune* defaults. These statements allow control over the name of the *Model 204* load module, the level of detail to which data is collected, the time intervals over which data is collected, the sampling rate, authorization to issue MODIFY commands, and more. For a list of the available statements, see "Configuration Statements for the Data Collector" on page 13.

  SIRTUNEI can have either fixed or variable format, and it can have any record length.

2.2 Version 1.5 or earlier of SirTune

The data collection portion of *SirTune* consists of a single load module called **SIRTUNE**.

2.2.1 Invoking the SIRTUNE module

To have *SirTune* collect data for a particular **ONLINE** or **BATCH204** job, you must do the following so that *SirTune* will run *Model 204* as a subtask of **SIRTUNE**, collecting polling data as required:

- Modify the JCL that invokes *Model 204* so that it invokes **SIRTUNE** instead. For example, to have *SirTune* monitor an **ONLINE** invoked with:

  ```
  //ONLINE   EXEC PGM=ONLINE,REGION=4096K,TIME=1440,
  //              PARM='LIBUFF=600,SYSOPT=155,NJBUFF=2'
  ```

  change the line to read:

  ```
  //ONLINE   EXEC PGM=SIRTUNE,REGION=4096K,TIME=1440,
  //              PARM='LIBUFF=600,SYSOPT=155,NJBUFF=2'
  ```

- Place the **SIRTUNE** load module into the same load library as the **ONLINE** or **BATCH204** load library, or concatenate the library containing the **SIRTUNE** load module with the *Model 204* load library.

SirTune data collection should have no significant impact on the performance of the **ONLINE** or **BATCH204** region.

If the *Model 204* load module that is being monitored with *SirTune* must run authorized, the SIRTUNE load module must be placed into an APF-authorized library.

If SIRTUNE is able to load *Model 204* but cannot sample for some reason (including unknown *Model 204* release, *SirTune* expiration, or operation on an unauthorized CPU), *Model 204* will still proceed. This lets you leave SIRTUNE in place in your JCL while a temporary problem is being solved.

2.2.2 The SIRTUNE DD statements

The SIRTUNE load module uses as many as three DD statements: the first (**SIRTUNED**) is required and specifies the dataset to receive the collected data; the other two are optional:

SIRTUNED See list item “SIRTUNED” on page 4.

SIRTUNEI See list item “SIRTUNEI” on page 4.

SIRTUNEO This optional DD receives informational SIRTUNE messages. If this DD is not specified, these messages go to the MVS job log. SIRTUNEO must have LRECL greater than or equal to 80.

2.2.3 JCL example

The following is an example of JCL that runs a BATCH204 job under SIRTUNE in an MVS environment.

```
//DAILY204 JOB  (0),CLASS=C,MSGCLASS=A
//BATCH204 EXEC PGM=SIRTUNE,REGION=4096K,
//              PARM='SYSOPT=209,LIBUFF=1000'
//STEPLIB  DD  DSN=M204.V410.LOADLIB,DISP=SHR
//         DD  DSN=SIRIUS.LOAD,DISP=SHR
//SIRTUNEI DD  *
PGM BATCH204
/*
//SIRTUNED DD  DSN=SIRTUNE.SAMPLE.DATA,DISP=SHR
//CCAPRINT DD  SYSOUT=*
//CCAAUDIT DD  SYSOUT=*
//CCASNAP  DD  SYSOUT=*
//CCATEMP  DD  UNIT=WORK,SPACE=(CYL,(40,0))
//CCASTAT  DD  DSN=M204.CCASTAT.DISP=SHR
//POPDATA  DD  DSN=LOCAL.PROD.POPDATA,DISP=SHR
//POPPROC  DD  DSN=LOCAL.PROD.POPPROC,DISP=SHR
//CCAIN    DD  *
SPCORE=20000,MAXBUF=1000,MINBUF=50,SERVSIZE=300000
     .....
//
```

CHAPTER 3 Collecting Data Under CMS: SIRTUNE

As of the integration of the *SirTune* data collector with the *Sirius Mods* (in *Sirius Mods* version 6.9), how you invoke *SirTune* depends on its version.

3.1 Versions of SirTune after 1.5

The data collection portion of *SirTune* is part of the *Sirius Mods* object. It also requires a load module called SIRTUNED, which runs in a separate service machine. The data collector becomes available once the *Sirius Mods* is link edited into the *Model 204* ONLINE module and once the SIRTUNED service machine is made available.

3.1.1 Invoking SirTune

To invoke *SirTune*, the EXEC that invokes the *Model 204* load module should do so directly:

```
M2Ø4CMS M2Ø4ONLN ...
```

This statement differs from that required for version 1.5 and earlier of *SirTune* (which is described in “Invoking the SIRTUNE module” on page 8).

SirTune also requires the presence of a virtual machine running the SIRTUNED load module (as described in “The SIRTUNED virtual machine” on page 9). Then, if *SirTune* is authorized for use at your site, the *SirTune* data collector will be initialized,

If you are upgrading from an earlier version of *SirTune*, no changes are necessary to any *SirTune* DDs you were using. However, if you specified SIRTUNEI configuration statements for the data collector, the PGM statement is ignored, since *SirTune* no longer loads the *Model 204* ONLINE or BATCH204 load module.

If you want to prevent the *SirTune* data collector from being initialized, do the following:

- Set the SIRTUNE parameter to 0.

  The SIRTUNE parameter, which controls whether the *SirTune* data collector is initialized at the start of a *Model 204* run, can be set to either of these values:

  0 Disables initialization of the integrated *SirTune* product for a particular run.
  1 Enables initialization (this is the default).

For example:

```
M204CMS M204ONLN ( SIRTUNE 0 ...
```

3.1.2 Optional SirTune DD name

SirTune has the optional DD described below, for which a FILEDEF can be added to M204FDEF EXEC or any other EXEC invoked before the ONLINE module.

Note: The SIRTUNEO DD used in earlier versions of *SirTune* is obsolete in versions of *SirTune* after 1.5.

SIRTUNEI This optional DD contains configuration statements that alter the *SirTune* defaults. These statements ("Configuration Statements for the Data Collector" on page 13) allow control over the name of the *Model 204* load module, the level of detail to which data is collected, the time intervals over which data is collected, the sampling rate, authorization to issue MODIFY commands, and more.

SIRTUNEI can have either fixed or variable format, and it can have any record length.

The following is a sample FILEDEF:

```
FILEDEF SIRTUNEI DISK SIRTUNE INPUT A
```

3.2 Version 1.5 or earlier of SirTune

The data collection portion of *SirTune* consists of a load module called SIRTUNE that runs in the *Model 204* ONLINE virtual machine or virtual machines and another load module called SIRTUNED which runs in a separate service machine.

3.2.1 Invoking the SIRTUNE module

To have *SirTune* collect data for a particular ONLINE machine, do the following:

1. Place the SIRTUNE load module on a minidisk accessible to the virtual machine running *Model 204*.

2. Modify the EXEC that invokes *Model 204* so that it invokes SIRTUNE instead.

   For example, to have SIRTUNE monitor an ONLINE that is invoked with

   ```
   M204CMS M204ONLN ( SYSOPT 155 LIBUFF 600
   ```

change the line to read

```
M204CMS SIRTUNE ( SYSOPT 155 LIBUFF 600
```

SIRTUNE will then invoke *Model 204*, collecting polling data as required.

SirTune data collection should have no significant impact on the performance of the ONLINE virtual machine.

If SIRTUNE is able to load *Model 204* but cannot sample for some reason (including unknown *Model 204* release, *SirTune* expiration, or operation on an unauthorized CPU), *Model 204* will still proceed. This allows leaving SIRTUNE in place in your EXECs while a temporary problem is being solved.

3.2.2 Optional SIRTUNE DD names

The SIRTUNE load module has some optional DDs. FILEDEFs can be added for these DDs to M204FDEF EXEC or any other EXEC invoked before SIRTUNE. The optional DDs are:

SIRTUNEI See "Optional SirTune DD name" on page 8.

SIRTUNEO This optional DD receives informational SIRTUNE messages. If this DD is not specified, SIRTUNE messages go to the virtual console. SIRTUNEO must have LRECL greater than or equal to 80.

The following is a sample FILEDEF:

```
FILEDEF SIRTUNEO FISK SIRTUNE LISTING A
```

3.3 The SIRTUNED virtual machine

SirTune requires the presence of a virtual machine running the SIRTUNED load module. It is recommended that this virtual machine be given the userid `SIRTUNED`. The installation tape contains a sample exec called SIRTUNED EXEC which can be used as PROFILE EXEC for this service machine.

This service machine communicates with all running ONLINE virtual machines running *SirTune*, saving their sample data to disk. Because of this, appropriate directory statements must be added to the CP directory to allow IUCV communications between the *Model 204* virtual machines and the SIRTUNED virtual machine. The easiest way to accomplish this is by adding the IUCV ALLOW directory statement for user SIRTUNED.

The sample SIRTUNED EXEC invokes the SIRTUNED MODULE as follows:

```
'SIRTUNED';
```

To authorize users to issue commands to SIRTUNED via SMSG, list the authorized users after the SIRTUNED command. The userids in the list can contain wildcard characters. For example,

```
'SIRTUNED SYSOPER MARGE*';
```

authorizes userid SYSOPER and any userid begining with the characters MARGE to issue a command to SIRTUNED via SMSG. For more information on using wildcards in this list, see “Wildcard Strings in SirTune and SIRTUNER Statements” on page 95.

SIRTUNED will use MSGNOH for its responses if it is authorized; otherwise it will use MSG.

It is generally not important to terminate the SIRTUNED service machine “cleanly” if the *Model 204* virtual machines running *SirTune* are themselves terminated cleanly. However, if it becomes necessary to terminate the SIRTUNED service machine while *Model 204* virtual machines are running, log onto SIRTUNED and issue any one of the following commands:

- QUIT
- END
- STOP
- SHUTDOWN

Note: When SIRTUNED terminates, data collection on all *Model 204* service machines running *SirTune* will be immediately terminated. They will, however, continue to run *Model 204* without interruption.

Every time an ONLINE running *SirTune* is brought up, *SirTune* attempts to establish an IUCV connection with SIRTUNED. If it is unable to do so, the ONLINE is not brought up. If a connection is established, SIRTUNED immediately invokes an EXEC called SIRTUNEF. This exec can then issue FILEDEFs or other commands as required.

After SIRTUNEF returns to SIRTUNED, SIRTUNED attempts to open the file that will contain the *SirTune* sample data. The default DDNAME used for the open is the userid of the *Model 204* service machine. The actual DDNAME can be modified with a *SirTune* statement (“Configuration Statements for the Data Collector” on page 13), and it is passed to SIRTUNEF.

A sample SIRTUNEF is provided on the installation tape and should be modified to suit installation requirements. SIRTUNEF is passed two parameters:

- The userid of the *Model 204* virtual machine
- The DDNAME to be used for the open (that is, the name to be used on a FILEDEF command)

If SIRTUNEF sets a non-zero return code, SIRTUNED will return an open error to SIRTUNED and close the IUCV connection, preventing the ONLINE from coming up. After SIRTUNEF returns to SIRTUNED, SIRTUNED attempts to open the appropriate DDNAME for output. If this open fails, an open error is reflected to *SirTune* and the IUCV connection is closed, preventing the ONLINE from coming up.

In general, it is sufficient to allow the sample datasets to reside on a CMS format minidisk. Although data can be sent to tape, delays in manual tape handling could result in hung *Model 204* virtual machines waiting for SIRTUNED to process a tape mount. Data can also be sent to OS format minidisks. To do this, however, SIRTUNED must be run under the *Model 204* CMS interface (M204CMS). To do this, change the line in SIRTUNED EXEC that reads

```
'SIRTUNED';
```

to read

```
'M2Ø4CMS SIRTUNED';
```

While it is somewhat more efficient to send sample data to an OS format minidisk than to a CMS format disk, this advantage is probably outweighed in most cases by the advantage of not having to preallocate space for each sample dataset.

The sample datasets must be variable format and should generally have a large block size (greater than 10000). If DCB information is not explicitly specified, the defaults selected by *SirTune* should be adequate for all but the most extreme cases. If a sample dataset fills up or a CMS minidisk becomes full, the virtual machine(s) running *SirTune* and associated with the full minidisk or dataset will simply stop collecting data for the duration of the run. 20 megabytes for each *SirTune* sample dataset should be sufficient for most shops, while 50 megabytes per dataset should be sufficient for almost any requirements.

Since the only cost of running out of space on SIRTUNED is the loss of some data, it's not worth spending a lot of time trying to size SIRTUNED exactly. Simply allocate the SIRTUNED minidisk at 20 megabytes per *Model 204* virtual machine for which data is to be collected (or less if disk space is tight), and adjust the size based on experience.

For more information on sizing SIRTUNED, see “Estimating SIRTUNED Size Requirements” on page 97. The sample SIRTUNEF EXEC on the installation tape is an example of how samples from several runs can be kept simultaneously available.

If *Model 204* ONLINE service machines are brought up (AUTOLOG'ed) automatically at system initialization and some of these virtual machines will run with *SirTune*, the SIRTUNED service machine must complete SIRTUNED initialization before any *SirTune* in a *Model 204* virtual machine attempts to establish an IUCV connection to SIRTUNED. Because SIRTUNED initialization is extremely quick, this can probably be guaranteed by placing an AUTOLOG command for SIRTUNED ahead of AUTOLOG commands for *Model 204* service machines in the exec(s) doing the AUTOLOG'ing. To be even more

certain, a `CP SLEEP x SEC` can be placed after the AUTOLOG of SIRTUNED to give SIRTUNED time to get through initialization, where *x* is some small number (1 is probably sufficient).

Using SIRTUNEA EXEC guarantees that the *Model 204* service machines won't be logged on until SIRTUNED has completed initialization. SIRTUNEA EXEC is invoked by SIRTUNED after it has completed initialization. If the AUTOLOG commands for the *Model 204* service machines are moved into SIRTUNEA EXEC, SIRTUNED is authorized to issue the appropriate AUTOLOGs, and an AUTOLOG command for SIRTUNED is added to the initialization exec, then SIRTUNED will automatically be logged on, and it will initialize and then bring up (AUTOLOG) the *Model 204* service machines.

CHAPTER 4 Configuration Statements for the Data Collector

An optional DD card (or FILEDEF under CMS) called SIRTUNEI allows users to specify characteristics of the reporting run and to customize which reports are requested and how they are ordered.

The configuration statements take the form of control cards. There can be only one *SirTune* configuration statement per line, there are no line continuations, and a line beginning with an asterisk character (*) is treated as a comment and ignored.

Available *SirTune* configuration statements are described in the subsections that follow. The minimum abbreviation allowed for each statement is indicated with uppercase characters, while optional characters are indicated with lowercase. Optional statement parameters are enclosed in brackets ([]). Alternatives are separated with a vertical bar (|).

4.1 ALLComp

This statement specifies that *SirTune* should collect compilation data for all procedures; it is the inverse of the PRECOMP statement (“ALLComp”).

The default for a single user run (NUSERS = 1) is that *SirTune* collects compilation data for all procedures. The default for a multi-user run (NUSERS > 1) is that *SirTune* collects compilation data only for pre-compiled procedures. It is unnecessary to specify ALLCOMP in a single-user run since it is the default.

If ALLCOMP is not used explicitly in a multi-user run, or if it is overridden by the PRECOMP statement in a single-user run, the report generator can produce line-by-line breakdowns for pre-compiled APSY procedures only. Using the ALLCOMP statement makes it possible for the reporting module to provide line-by-line breakdowns for all User Language procedures executed in the job.

Specifying the ALLCOMP statement can significantly increase the size of the sample dataset, and in extreme cases, it can adversely affect the performance of an ONLINE. Because of this, it is generally recommended that you use only the ALLCOMP option when running *SirTune* against a single-user batch job, or when running multi-user jobs that do a lot of processing in non-APSY or non-pre-compiled procedures.

Note: *SirTune* cannot provide line-by-line breakdowns for User Language programs not contained in procedures, including in-line procedures in CCAIN, IODEV3, or BATCH2 input streams, or those entered at a terminal without use of the editor.

4.2 AUTHorize userid1 [userid2] ...

This statement specifies the userids from whom *SirTune* should accept MODIFY commands (under MVS) or SMSG commands (under CMS). The default is that MODIFY or SMSG commands to *SirTune* are not accepted from any userid.

Since MVS provides no information about the user issuing a MODIFY statement, the only AUTHORIZE statement that makes any sense under MVS is:

```
AUTHORIZE *
```

This indicates that *SirTune* will accept MODIFY commands from any user that has the privilege to issue a MODIFY command (either an operator at a console or a user in SDSF).

Under CMS, an arbitrary number of userids can be specified, and the userids can contain wildcard characters. For example:

```
AUTHORIZE BART HOMER*
```

The statement above indicates that *SirTune* should accept SMSG commands from userid `BART` and from any userid that starts with the characters `H O M E R`. For more information on wildcards, see "Wildcard Strings in SirTune and SIRTUNER Statements" on page 95.

SIRTUNEI can contain an arbitrary number of AUTHORIZE statements. The AUTHORIZE statements are cumulative, so the userids specified on an AUTHORIZE statement are added to the list of userids specified by all previous AUTHORIZE statements.

For more information on the commands that can be issued via MODIFY or SMSG by authorized users, see "MODIFY and SMSG commands" on page 27.

4.3 CMSout vmid [ddname]

This statement has meaning only under CMS. It specifies the userid of the virtual machine (*vmid*) running the SIRTUNED load module, and it specifies the DD name (*ddname*) that virtual machine is to use for the sample dataset. The default for *vmid* is `SIRTUNED`, and the default for *ddname* is the userid of the virtual machine running *SirTune*.

For example, this statement indicates that there is a SIRTUNED service machine with userid `HOHO` that should use DD name `MARGE` for the sample dataset:

```
CMSOUT HOHO MARGE
```

The CMSOUT statement makes it possible to run multiple SIRTUNED service machines, because it allows *SirTune* in each *Model 204* service machine to indicate the userid of its SIRTUNED service machine. This is not a generally recommended mode of operation, however, because of the extra overhead of running multiple virtual machines.

4.4 COLLect state [extra_data]

This statement makes it possible to increase the quantity of data collected by *SirTune*. While collecting extra data increases the size of the sample dataset, it also makes it possible to produce extended reports.

The default for COLLECT in multi-user runs is `COLLECT RUNG`, which means that data is only collecting for running users (users consuming CPU), and no extra data is collected. The default for COLLECT in single-user runs is `COLLECT ALLN BLKU DISKIO`, which means that the only user for which data is not collected is waiting PSTs. Data is always collected for running users, regardless of the COLLECT statements in SIRTUNEI.

state is a list of one or more blank-delimited user states. *SirTune* will collect sample data for a user in any of the states indicated by COLLECT statements at the moment a sample is collected.

For example, if the following is specified, *SirTune* will collect sample data for any user that is swapping into or out of a server at the instant a sample is collected:

```
COLLECT SWPG
```

To be able to produce a report for a particular state, data must be collected for that state with *SirTune*. For example, if you want to produce a report for STATE SWPG, you can ensure the appropriate data will be collected by *SirTune* by placing this statement in SIRTUNEI:

```
COLLECT SWPG
```

For a discussion of available states and their meanings, see “Model 204 States” on page 75.

In addition to requesting that sample data be collected for users in specific states, it is also possible to request extra data to isolate the cause of certain wait types. The only valid values for *extra_data* are `DISKIO` and/or `CFR`.

Specifying `DISKIO` as a parameter on a COLLECT statement indicates that *SirTune* should collect data for every wait on *Model 204* disk I/O. This data allows the reporting module to produce the DISKIO reports. These reports allow breakdowns of waits on *Model 204* disk I/O by file, table, and groups of pages within tables.

Specifying DISKIO on a collect statement implies the BLKIN parameter. That is, this statement:

```
COLLECT DISKIO BLKIN
```

is functionally equivalent to this:

```
COLLECT DISKIO
```

Specifying CFR as a parameter on a COLLECT statement indicates that *SirTune* should collect data for users with “Critical File Resource” activity. Critical file resources are used to provide concurrency control among updating and retrieving operations on the same file. Contention on critical file resources can exacerbate performance problems, sometimes dramatically.

The CFR data allows the reporting module to produce the CFRROOT, STATE CFRH???, and STATE CFRB??? reports. These reports allow breakdowns of:

- The root causes of critical file resource waits
- Activities that result in the holding of critical file resources

Since critical file resources are used to provide multi-user concurrency control, the CFR parameter is meaningless in a single-user run.

COLLECT statements are cumulative. That is, this sequence:

```
COLLECT SWPGI
COLLECT BLKIN
COLLECT BLKON
```

is functionally equivalent to this:

```
COLLECT SWPGI BLKIN BLKON
```

and it results in data being collected for all users in state SWPGI, BLKIN, or BLKON. There is no limit to the number of COLLECT statements in SIRTUNEI.

The first COLLECT statement in SIRTUNEI for a single-user run is treated as an override to the single-user default. That is, if the following is the first COLLECT statement in SIRTUNEI for a single-user run:

```
COLLECT RUNG
```

it would turn off data collection for all other user states in the single-dser default and for DISKIO data.

4.5 EXClude [start_time end_time] [days_of_week]

This statement makes it possible to limit the times in which sample data is collected. Note that compilation data is always collected, whether or not sample data is being collected.

start_time and *end_time* can have the format HH, HH:MM, or HH:MM:SS, where HH represents an hour of the day, MM represents minutes, and SS represents seconds. The maximum allowable time is 24 or 24:00 or 24:00:00, which corresponds to midnight.

For example, the following indicates that no sample data should be collected from midnight through 8 AM every day of the week:

```
EXCLUDE 0 8
```

The following indicates that no sample data should be collected from 11:30 AM through 1:30 PM every day of the week:

```
EXCLUDE 11:30 13:30
```

The following indicates that no sample data should be collected from 30 seconds before 12 noon through 50 seconds after 1 PM every day of the week:

```
EXCLUDE 11:59:30 13:00:50
```

If *start_time* is greater than *end_time*, the times are assumed to wrap. For example, the following means that no sample data should be collected between 11 PM and 1 AM:

```
EXCLUDE 23:00 1:00
```

days_of_week is a blank-delimited list of days of the week. The following are valid days of the week, with the minimum abbreviations in uppercase:

- SUNday
- MONday
- TUEsday
- WEDnesday
- THUrsday
- FRIday
- SATurday

For example, the following indicates that no sample data should be collected all day Sundays and Saturdays:

```
EXCLUDE SUNDAY SAT
```

The following indicates that no sample data should be collected from 6:30 PM through midnight on Mondays through Fridays:

```
EXCLUDE 18:30 24:00 MOND TUES WEDN THU FRIDAY
```

In addition to the days of the week, the following groupings are also available in a *days_of_week* list:

- WEEKDay = Monday, Tuesday, Wednesday, Thursday, Friday
- WEEKEnd = Saturday, Sunday
- ALL = Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

For example, the following indicates that no sample data should be collected all day Saturday through Sunday:

```
EXCLUDE WEEKE
```

The following indicates that no sample data should be collected on any day of the week:

```
EXCLUDE ALL
```

The following indicates that no sample data should be collected between 10 AM and 12 noon on Sunday, Saturday, and Monday:

```
EXCLUDE 10 12 WEEKEND MONDAY
```

The following indicates that no sample data should be collected between 11 PM and midnight on all days of the week:

```
EXCLUDE 23:00 24:00 ALL
```

Note that the statement above is identical to this:

```
EXCLUDE 23:00 24:00
```

If no INCLUDE or EXCLUDE statements occur in SIRTUNEI, sample data is collected as if INCLUDE ALL had been specified.

The order of INCLUDE and EXCLUDE statements in SIRTUNEI is important: If the first INCLUDE or EXCLUDE statement in SIRTUNEI is `EXCLUDE`, it is treated as if it were preceded by an `INCLUDE ALL`. That is, data will only not be collected over intervals explicitly indicated by EXCLUDE statements.

For example, if this is the only INCLUDE or EXCLUDE statement in SIRTUNEI:

```
EXCLUDE 0 9
```

sample data will be collected between 9 AM and midnight every day of the week. No data will be collected at other times. If SIRTUNEI contains the following sequence, data should be collected on Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday:

```
EXCLUDE WEEKEND
INCLUDE SATURDAY
```

If SIRTUNEI contains the following sequence, data should be collected on every weekday from 8 AM to 5 PM and on Fridays from 10 PM to midnight:

```
EXCLUDE WEEKEND
EXCLUDE 0 8
EXCLUDE 17 24
INCLUDE 22 24 FRIDAY
```

Note that there is no harm in EXCLUDE'ing a range that has already been excluded. For example, the following

```
EXCLUDE 0:00 7:00
EXCLUDE 4:00 10:00
```

is the same as

```
INCLUDE 0:00 10:00
```

The INCLUDE/EXCLUDE statements are only honored if sampling is set to AUTO (automatic) mode, either by default or via a SAMPLE statement or a MODIFY or SMSG SAMPLE command. When sampling is in manual mode, INCLUDE/EXCLUDE statements have no effect.

Generally, you should use the INCLUDE/EXCLUDE statements to avoid collecting data for time intervals in which there is nothing interesting likely to be happening, that is, periods of low activity. The only real cost of this is the disk space and the minor amount of CPU consumed collecting this sample data.

If you are only interested in collecting data at peak hours, say 10 AM to 11:30 AM and 1:30 PM to 4 PM on weekdays, simply put the following in SIRTUNEI:

```
INCLUDE 10 5 WEEKDAY
EXCLUDE 11:30 13:30
```

4.6 INClude [start_time end_time] [days_of_week]

This statement makes it possible to limit the times in which sample data is collected. Note that compilation data is always collected, whether or not sample data is being collected.

start_time and *end_time* can have the format HH, HH:MM, or HH:MM:SS where HH represents an hour of the day, MM represents minutes, and SS represents seconds.

The maximum allowable time is 24 or 24:00 or 24:00:00, which corresponds to midnight. For example, the following indicates that sample data should be collected from 8 AM through 6 PM every day of the week:

```
INCLUDE 8 18
```

The following indicates that sample data should be collected from 8:30 AM through 5:30 PM every day of the week:

```
INCLUDE 8:30 17:30
```

The following indicates that sample data should be collected from 10 seconds before 9 AM through 10 seconds after 5 PM every day of the week:

```
INCLUDE 8:59:50 17:00:10
```

If *start_time* is greater than *end_time*, the times are assumed to wrap. For example, the following means that sample data should be collected between 11 PM and 1 AM:

```
INCLUDE 23:00 1:00
```

days_of_week is a blank-delimited list of days of the week. The following are valid days of the week, with the minimum abbreviations in uppercase:

- SUNday
- MONday
- TUEsday
- WEDnesday
- THUrsday
- FRIday
- SATurday

For example, indicates that sample data should be collected all day Mondays and Tuesdays:

```
INCLUDE MONDAY TUE
```

The following indicates that sample data should be collected from 8:45 AM through 5:45 PM on Mondays through Fridays:

```
INCLUDE 8:45 17:45 MOND TUES WEDN THU FRIDAY
```

In addition to the days of the week, the following groupings are also available in a *days_of_week* list:

- WEEKDay = Monday, Tuesday, Wednesday, Thursday, Friday
- WEEKEnd = Saturday, Sunday
- ALL = Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

For example, the following indicates that sample data should be collected all day Monday through Friday:

```
INCLUDE WEEKD
```

The following indicates that sample data should be collected all day on every day of the week:

```
INCLUDE ALL
```

The following indicates that sample data should be collected between 10 AM and 12 noon on Sunday, Saturday, and Monday:

```
INCLUDE 10 12 WEEKEND MONDAY
```

The following indicates that sample data should be collected between 11 PM and midnight on all days of the week:

```
INCLUDE 23:00 24:00 ALL
```

Note that the above statement is identical to this:

```
INCLUDE 23:00 24:00
```

If no INCLUDE or EXCLUDE statements occur in SIRTUNEI, sample data is collected as if INCLUDE ALL had been specified.

The order of INCLUDE and EXCLUDE statements in SIRTUNEI is important: If the first INCLUDE or EXCLUDE statement in SIRTUNEI is `INCLUDE`, it is treated as if it were preceded by an `EXCLUDE ALL` (that is, data will only be collected over intervals explicitly indicated by INCLUDE statements).

For example, if this is the only INCLUDE or EXCLUDE statement in SIRTUNEI:

```
INCLUDE 9 17
```

sample data should be collected between 9 AM and 5 PM every day of the week. No data should be collected at other times. If SIRTUNEI contains the following sequence:

```
INCLUDE WEEKDAY
EXCLUDE TUESDAY THURSDAY
```

data should be collected on Monday, Wednesday, and Friday. If SIRTUNEI contains the following sequence:

```
INCLUDE 8 17 TUESDAY
EXCLUDE 11 14
INCLUDE 12 13 TUESDAY
```

data should be collected on Tuesdays from 8 AM to 11 AM, 12 noon to 1 PM, and 2 PM to 5 PM.

Note: There is no harm in INCLUDE'ing a range that has already been included. For example, the following range

```
INCLUDE 9:00 13:00
INCLUDE 12:00 17:00
```

is the same as

```
INCLUDE 9:00 17:00
```

The INCLUDE/EXCLUDE statements are only honored if sampling is set to AUTO (automatic) mode, either by default, or via a SAMPLE statement or a MODIFY or SMSG SAMPLE command. When sampling is in manual mode, INCLUDE/EXCLUDE statements have no effect.

Generally, you should use the INCLUDE/EXCLUDE statements to avoid collecting data for time intervals in which there is nothing interesting likely to be happening, that is, periods of low activity. The only real cost of collecting data in periods of low activity is wasted disk space and a minor amount of CPU consumed collecting the sample data.

If you are only interested in collecting data at peak hours, say 10 AM to 11:30 AM and 1:30 PM to 4 PM on weekdays, simply put the following statements into SIRTUNEI:

```
INCLUDE 10 11:30 WEEKDAY
INCLUDE 1:30 4 WEEKDAY
```

4.7 INTerval num_sec

This statement is used to set the sampling rate, thus making it possible to limit the amount of sample data collected. The default for *num_sec* is 1, which means that sample data is collected at one-second intervals.

The values specified for *num_sec* have a maximum resolution of 1/100 of a second so that

```
INTERVAL 3.1415929
```

is identical to

```
INTERVAL 3.14
```

This also means that the lowest legal value for *num_sec* is 0.01.

The objective in setting the sampling interval should be to collect sufficient data to produce statistically significant results, without collecting unnecessarily huge amounts of data. Setting the sampling interval extremely low can also produce biased results. Generally, sampling intervals greater than 0.1 seconds should avoid any biasing problems.

A reasonable rule of thumb is that 10,000 samples are sufficient to produce accurate reports. Thus to collect data on a batch job that generally takes half an hour to complete, samples should be collected every 1800/10000 seconds. To accomplish this, place the following statement into SIRTUNEI:

```
INTERVAL Ø.18
```

If data is to be collected over 8 hours of a production ONLINE run, the default sampling interval of 1 second would result in 8*3600 or 28,800 samples being collected. In this case, it would not be unreasonable to place the following statement into SIRTUNEI to reduce the number of samples collected to 14,400:

```
INTERVAL 2
```

4.8 MIXed

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement indicates that *SirTune* should issue all messages in mixed-case. The default is to issue messages in mixed-case except on Japanese operating systems. On these systems, the default is to issue all messages in uppercase only. This option should be used on Japanese operating systems when mixed-case messages are desired.

4.9 NOSeq

This statement indicates that *SirTune* should not consider characters in columns 73 through 80 of SIRTUNEI as part of its input. If SIRTUNEI has RECFM=F and LRECL=80, *SirTune* ordinarily assumes that characters in columns 73 through 80 are sequence numbers, and therefore it ignores them. If SIRTUNEI has any other format, *SirTune* considers all characters in the input record as part of the *SirTune* statements.

4.10 PGM pgm_name

Effective in version 1.5 of *SirTune* and lower, this statement is ignored in versions after 1.5.

The PGM statement makes it possible to specify the name of the *Model 204* ONLINE or BATCH204 load module to be loaded by the SIRTUNE module. The default for *pgm_name* is `ONLINE` under MVS, and it is `M204ONLN` under CMS.

Thus, if SIRTUNE is to collect data for a job that should be run with BATCH204, put the following statement into SIRTUNEI:

```
PGM BATCH204
```

If the name of the load module for which SIRTUNE is to collect data is ONLINE22, put the following statement into SIRTUNEI:

```
PGM ONLINE22
```

4.11 PREComp

This statement specifies that *SirTune* should only collect compilation data for pre-compiled APSY procedures; it is the inverse of the ALLCOMP command (“ALLComp” on page 13).

The default for a single-user run (NUSERS = 1) is that *SirTune* collects compilation data for all procedures. The default for a multi-user run (NUSERS > 1) is that *SirTune* collects compilation data only for pre-compiled procedures. It is unnecessary to specify PRECOMP in a multi-user run, since it is the default.

When PRECOMP is in effect, either because of an explicit PRECOMP statement or as the result of multi-user run defaults, the reporting module can produce line-by-line breakdowns for pre-compiled APSY procedures only.

4.12 SAMPle ON | OFF | AUTO

This statement places *SirTune* in manual sampling mode. In this mode, *SirTune* ignores INCLUDE and EXCLUDE statements until a `SAMPLE AUTO` command is issued via a MODIFY or SMSG command issued by an authorized user. The default for this statement is SAMPLE AUTO, which places *SirTune* under the control of INCLUDE and EXCLUDE statements.

For example, to have *SirTune* come up so that no sample data is collected until a user issues a (MODIFY or SMSG) SAMPLE command to turn sampling off or to put sampling in AUTO mode, simply place the following statement into SIRTUNEI:

```
SAMPLE OFF
```

To have *SirTune* come up so that sample data is collected until a user issues a MODIFY or SMSG SAMPLE command, place the following statement into SIRTUNEI:

```
SAMPLE ON
```

Note: If SAMPLE OFF is specified in SIRTUNEI and there are no AUTHORIZE statements in SIRTUNEI, it will be impossible to collect any sample data for the course of the run, because no user will have the authority to issue the MODIFY or SMSG SAMPLE command to set sampling to ON or to AUTO.

4.13 UPper

This statement indicates that *SirTune* should issue all messages in uppercase only. The default is to issue messages in mixed-case, except on Japanese operating systems. On these systems the default is to issue all messages in uppercase only.

This option should be used on systems where mixed-case messages might not be displayed correctly on terminals or printers.

CHAPTER 5 MODIFY and SMSG commands

Users outside the *Model 204* or SIRTUNE/*Model 204* address space can be authorized to issue certain commands to *SirTune* while the ONLINE or BATCH204 job is running. Users can be authorized to do this with *SirTune*'s AUTHORIZE command. For more information on the AUTHORIZE command, see "Configuration Statements for the Data Collector" on page 13.

Under MVS, these commands can be issued via the MODIFY operator command. This command can be issued at an operator console or under SDSF or an equivalent virtual console system. For example, to issue the STATUS command to *SirTune* running under job PRODONLN under SDSF's LOG screen, you can simply enter

```
/MODIFY PRODONLN,STATUS
```

or

```
/F PRODONLN,STATUS
```

Responses to MODIFY commands go to the system log and should be viewable under SDSF.

Under CMS, these commands can be issued via the SMSG CP command. For example, to issue the STATUS command to *SirTune* running on a virtual machine named PRODONLN, you can enter

```
SMSG PRODONLN STATUS
```

or

```
CP SMSG PRODONLN STATUS
```

Responses to SMSG commands are sent via MSGNOH, if the *SirTune*/*Model 204* service machine is authorized to use MSGNOH, and they are sent via MSG otherwise.

Available MODIFY/SMSG commands are listed here. Optional statement parameters are enclosed in brackets ([]). Alternatives are separated with a vertical line (|).

5.1 BUMP user_num

This command requests that *SirTune* issue the equivalent of the *Model 204* BUMP command against the indicated user number. This command is useful if a high priority user is looping in an ONLINE, making it impossible for anyone else to do anything. To “bump” user number 18, issue this command:

```
BUMP 18
```

Unlike the *Model 204* BUMP command, the *SirTune* BUMP command accepts only a single user number, and it does not accept userids or file names. To determine the user number of the running user, issue *SirTune*'s MONITOR command. It is recommended that the MONITOR command be issued several times to ensure that a single user is indeed looping and that a performance problem is not simply the result of excess demand.

If a loop situation is caused by a *Model 204* bug, it is possible that a BUMP command will fail to force the looping user off the system. In this case, it might be necessary to issue the RESTART command to break the loop.

5.2 CLOSE

This command requests that *SirTune* close the sample dataset. This makes it possible to run the report generator against a sample data set that is still being updated by *SirTune*. This command has no effect on sampling activity in the *Model 204* address space.

5.3 MONITOR

This command requests that *SirTune* return information about what is currently happening in *Model 204* This command is especially useful if *Model 204* appears to be hung or looping and, hence, it is impossible to log on to *Model 204* to determine the cause of the problem.

Information is returned for the main *Model 204* task and any subtasks (if the MP/204 feature is being used). The information returned indicates whether each task is running or waiting (corresponding to looping and hung situations respectively) and where in the *Model 204* load module the task is running or waiting. This latter piece of information is most useful to *Model 204* internals experts.

If any *Model 204* task is indicated as running, information is also provided on the user number and userid of the running user, the current activity (evaluating, compiling, etc.), and the name of the procedure that is currently running. This information can be used to determine whether a BUMP or RESTART command should be issued.

5.4 RESTART abend_code USER user_num | TASK task_num

This command requests that *SirTune* issue the equivalent of a user abend in *Model 204* for the indicated user or task. This command should be used only as a last resort when a *Model 204* ONLINE is looping or hung and all efforts to clear up the situation have failed (see the BUMP command). In this situation, the only options left are to cancel (FORCE under CMS) the run or to issue *SirTune*'s RESTART command. The RESTART command is preferable because:

- The ONLINE might intercept the abend, possibly take a snap dump, and then continue running.
- Even if the ONLINE does not continue, it might at least intercept the abend and terminate "cleanly," preventing files from being broken and eliminating the need to run recovery. Cancelling or forcing the ONLINE ensure that any files with active updating transactions will be left broken and that recovery will have to be run before the ONLINE can be used again.
- The RESTART command will generally result in a CCASNAP being taken rather than (or in addition to) a SYSMDUMP, SYSUDUMP, or VMDUMP. CCASNAPs are generally easier to deal with than other types of dumps.

The *abend_code* value must be a 1- to 3-digit hexadecimal code that indicates the abend code to be used for the artificially generated abend. Under CMS the abend code must be between 0C1 and 0CF. Any dumps will indicate the specified abend code. It is important to inform support personnel examining the dump that the abend code was artificially generated by *SirTune* and that the situation was actually a hung or looping ONLINE.

To prevent accidentally terminating a user or an ONLINE, a user number can be specified on the restart command. When a user number is specified on the RESTART command, no simulated abend will occur if the indicated user is not running in the ONLINE. For example, the following RESTART simulates a 0C4 abend if user 23 is running. If user 23 is not running, no abend will be simulated.

```
RESTART 0C4 USER 23
```

If no user is running (a hung ONLINE), it might be necessary to issue the RESTART command with a task number instead. Unless running MP/204, the task number must be specified as 0. If MP/204 is running, the task number must be 0 (for the maintask) or the subtask number (from 1 to NMPSUBS) to be restarted. For example, the following RESTART simulates a 0A9 abend on the *Model 204* maintask:

```
RESTART 0A9 TASK 0
```

Note: It is almost always preferable to specify a user number on the RESTART command rather than a task number.

5.5 SAMPLE ON | OFF | AUTO

This command either places sampling back into automatic mode (SAMPLE AUTO), where sampling is controlled by INCLUDE/EXCLUDE statements in SIRTUNEI, or it initiates (SAMPLE ON) or terminates (SAMPLE OFF) collection of sample data. This command has no effect on the collection of compilation data: Compilation data is collected regardless of the sampling state.

For example, to immediately start collecting data, the following command should be issued:

```
SAMPLE ON
```

To immediately stop collecting data, issue:

```
SAMPLE OFF
```

After a SAMPLE ON or SAMPLE OFF command is issued, *SirTune* is in “manual” sampling mode.

5.6 STATUS

This command requests the current sampling status of *SirTune*. *SirTune*'s response to this command indicates the current sampling mode (AUTO or MANUAL), the current sampling state (ON or OFF), and the number of samples collected to this point.

5.7 STOP

This command requests that *SirTune* stop collecting both sampling and compilation data and that it close the sample dataset. After this command is issued, sampling cannot be restarted for the run. Since the sample dataset is closed by a STOP command, it is possible to use this dataset to generate reports after a STOP command, even while the ONLINE/BATCH204 job continues to run.

CHAPTER 6 Generating Reports

As of version 7.2 of the *Sirius Mods*, the report generation portion of *SirTune* consists of two User Language programs which are distributed in the SIRIUS procedure file as part of *UL/SPF*.

For versions of the *Sirius Mods* prior to 7.2, the report generation portion of *SirTune* consists of a single load module called SIRTUNER. For information about running SIRTUNER, see "Generating reports prior to Sirius Mods 7.2: MVS" on page 33 and "Generating reports prior to Sirius Mods 7.2: CMS" on page 34.

The User Language reporting programs of *SirTune* are:

SIRTUNEREPORT The "outer" *SirTune* report program.

SHARED_REPORT Report methods used by SIRTUNEREPORT.

To run SIRTUNEREPORT from a procedure file other than SIRIUS, move both these procedures to the new file, and update the single INCLUDE statement in SIRTUNEREPORT. A few UTABLE and environmental settings are executed at the top of SIRTUNEREPORT. These can be adjusted for performance or functional reasons at the client site.

SIRTUNEREPORT requires two input files, and it accepts an optional output file. These files, described below, may be specified as DD cards or ALLOCATE statements on the job running SIRTUNEREPORT:

TUNERPTI Input cards that specify the *SirTune* reports and other run characteristics.

This file may contain older style input, with REPORT, RANGE, RESOLUTION, and other statements, or it may contain newer-style input in XML format. If old-style card images are input, a subroutine in SIRTUNEREPORT converts them to XML. For detailed information about this input, see "Configuring the Report Generator" on page 37.

TUNERPTI is a required file.

The file format of TUNERPTI is not critical: Since input lines should be shorter than 80 characters, a fixed 80-character file will work just fine.

TUNERPTD This is the input data to SIRTUNEREPORT.

TUNERPTD is a required file.

This is the same file as SIRTUNED in the *SirTune* data collection run.

Sample datasets from multiple runs may be concatenated. In fact, sample datasets from multiple runs may even be concatenated sequentially into a single dataset (with IEBGENER under MVS, or COPYFILE or an equivalent under CMS) and provided as a single dataset to SIRTUNEREPORT.

When multiple sample datasets are concatenated in any way, the DATASET statement ("DATaset list" on page 40) must be provided in TUNERPTI to have TUNERPTR use any but the first sample dataset in the concatenation.

TUNERPTO This is the output file for the SIRTUNEREPORT reports.

The User Language is currently set to output 72 characters per line, so a fixed or fixed-block file with 80-character records is recommended.

TUNERPTO is an optional file. If it is not present, report output will be redirected to standard output; generally, CCAPRINT.

The names of these files are hard-coded in SIRTUNEREPORT, but they may be changed if you run SIRTUNEREPORT from another procedure file and decide to customize it. The names are chosen to differentiate them from the SIRTUNEx names used during *SirTune* data collection.

Generally, SIRTUNEREPORT is run as a standalone batch job, though it is perfectly acceptable, if perhaps a little resource-intensive, to dynamically allocate the TUNERPTx files and run it from inside an Online.

A basic TUNERPTR job under MVS would look like this:

```
//TUNE JOB (0,0),CLASS=A,MSGCLASS=X
//TUNERPT  EXEC PGM=BATCH204,REGION=4096K,
//              PARM='SYSOPT=134,LIBUFF=1000,LOBUFF=1000',TIME=250
//CCASTAT  DD DSN=M204.CCASTAT,DISP=SHR
//CCATEMP  DD DSN=M204.TEMP1,DISP=SHR
//CCAJRNL  DD DSN=M204.CCAJRNL,DISP=SHR
//SIRIUS   DD DSN=ULSPF700.SIRIUS,DISP=SHR
//TUNERPTI DD DSN=SIRTUNER.REPORT.INPUT,DISP=SHR
//TUNERPTD DD DSN=SIRTUNER.DATA.INPUT,DISP=SHR
//TUNERPTO DD DSN=SIRTUNER.REPORT.OUTPUT,DISP=SHR
//CCAPRINT DD SYSOUT=*,OUTLIM=50000
//CCAIN    DD *
NFILES=5,NDCBS=5,NDIR=5,MAXBUF=50,NGROUP=5,NORQS=4,LIBUFF=132,     X
LOBUFF=320,LPDLST=2400,LQTBL=1350,LSTBL=2000,LNTBL=1200
LOGON me
  mepassword
  O SIRIUS
  INCLUDE SIRTUNEREPORT
/*
//
```

Under CMS, a similar Batch204 EXEC can be used to invoke SIRTUNEREPORT.

Compatibility notes:

- As of release 7.2 of the Sirius mods, *SirTune* reports can only be generated using a User Language program invoking a call to the `sirtuneReport` method of the Dataset class, as implemented in the SIRTUNEREPORT User Language program described in this chapter.
- Due to format changes in the sample dataset, only releases 1.6 and later of SIRTUNER can process a SirTune version 7.0 and later sample dataset.
- Due to format changes in the sample dataset in SirTune version 1.5, earlier releases of SIRTUNER cannot process a SirTune 1.5 or higher sample dataset.

6.1 Generating reports prior to Sirius Mods 7.2: MVS

For pre-7.2 versions of the *Sirius Mods*, the report generation portion of *SirTune* consists of a single load module called **SIRTUNER**. To have SIRTUNER generate a report for a particular ONLINE or BATCH204 job, code JCL that invokes SIRTUNER with appropriate DD cards.

Note: Due to format changes in the sample dataset, only releases 1.6 and later of SIRTUNER can process a *SirTune* version 7.0 and later sample dataset.

Note: Due to format changes in the sample dataset in *SirTune* version 1.5, earlier releases of SIRTUNER cannot process a *SirTune* 1.5 or higher sample dataset.

The DD names used by SIRTUNER are:

SIRTUNED This DD should point to a sample dataset generated by *SirTune*. It corresponds to the SIRTUNED DD for the data collection part of *SirTune*.

Sample datasets from multiple runs may be concatenated. In fact, sample datasets from multiple runs may even be concatenated sequentially into a single dataset (with IEBGENER or an equivalent) and provided as a single dataset to SIRTUNER.

When multiple sample datasets are concatenated in any way, the DATASET statement (“DATaset list” on page 40) must be provided in SIRTUNEI to have SIRTUNER use any but the first sample dataset in the concatenation.

SIRTUNEO This DD is the dataset that should receive SIRTUNER error messages or reports. It must have an LRECL greater than or equal to 80.

SIRTUNEI This optional DD contains SIRTUNER statements that alter the SIRTUNER defaults. These statements allow control over which reports are produced and the time interval over which sample data is to be summarized. For a list of available statements, see "Configuring the Report Generator" on page 37.

SIRTUNEI can have any format (RECFM=F or RECFM=V).

The following is an example of JCL used to generate a report with SIRTUNER:

```
//TUNER    JOB  (Ø),CLASS=C,MSGCLASS=A
//SIRTUNER EXEC PGM=SIRTUNER,REGION=8M
//STEPLIB  DD  DSN=SIRIUS.LOAD,DISP=SHR
//SIRTUNEI DD  *
REPORT STATE WDISK EVAL
/*
//SIRTUNED DD  DSN=SIRTUNE.SAMPLE.DATA,DISP=SHR
//SIRTUNEO DD  SYSOUT=*
//
```

6.2 Generating reports prior to Sirius Mods 7.2: CMS

For pre-7.2 versions of the *Sirius Mods*, The report generation portion of *SirTune* consists of a single load module: **SIRTUNER**. To have SIRTUNER generate a report for a particular ONLINE or BATCH204 job, code an EXEC that invokes SIRTUNER with appropriate FILEDEF commands.

Note: Due to format changes in the sample dataset, only releases 1.6 and later of SIRTUNER can process a *SirTune* version 7.0 and later sample dataset.

Note: Due to format changes in the sample dataset in *SirTune* version 1.5, earlier releases of SIRTUNER cannot process a *SirTune* 1.5 or higher sample dataset.

SIRTUNER can run under CMS or CMS/XA. Under CMS/XA, most of the storage used by SIRTUNER is "above the line," so virtual storage constraints should not be an issue.

The DD names used by SIRTUNER are:

SIRTUNED This DD should point to a sample dataset generated by *SirTune*. It corresponds to the SIRTUNED DD for the data collection part of *SirTune*.

Sample datasets from multiple runs may be concatenated. In fact, sample datasets from multiple runs can be concatenated sequentially into a single dataset (with COPYFILE or an equivalent) and provided as a single dataset to SIRTUNER.

When multiple sample datasets are concatenated in any way, the DATASET statement (“DATaset list” on page 40) must be specified in SIRTUNEI to have SIRTUNER use any but the first sample dataset in the concatenation.

SIRTUNEO This DD is the dataset that should receive SIRTUNER error message or reports. It must have an LRECL greater than or equal to 80.

SIRTUNEI This optional DD contains SIRTUNER statements that alter the SIRTUNER defaults. These statements allow control over which reports are produced and the time interval over which sample data is to be summarized. For a list of available statements, see “Configuring the Report Generator” on page 37.

SIRTUNEI must have fixed format, that is, RECFM=F or RECFM=FB.

The following sample EXEC can be used to generate a report with SIRTUNER:

```
/* */
address command
"FILEDEF SIRTUNEI DISK SIRTUNER INPUT A"
"FILEDEF SIRTUNEO DISK SIRTUNER LISTING A"
"FILEDEF SIRTUNED DISK PRODONLN DATA W",
   "(LRECL 512 BLOCK 4000 RECFM VB"

"SIRTUNER"

exit rc
```

CHAPTER 7 Configuring the Report Generator

An optional DD card allows the reporting run to be customized. For *SirTune* 7.2, using the SIRTUNEREPORT User Language program to generate reports, this DD card is named `TUNERPTI`. For *SirTune* versions 6.9 and older, using the SIRTUNER module, the DD card is named `SIRTUNEI`.

This input card must have fixed format, and it must use report configuration parameters specified in either of the following types of input format to define the characteristics of the reporting run:

- Control-card statements

  Report configuration parameters are specified in a set of statements (only one per line). There are no line continuations, and a line beginning with an asterisk character (`*`) is treated as a comment (that is, ignored).

  The individual statements are described in "Configuration parameters" on page 38.

  An example of control-card statement input follows:

```
RANGE 9:30 12:30
RESOLUTION 50
REPORT CFRROOT
REPORT STATE CFRBANY WHAT
MAXD 10
REPORT STATE RUNG QUAD
REPORT STATE RUNG EVALI EVAL CHUNK 32
REPORT STATE RUNG CHUNK 4 CHUNK 400 CHUNK 4000
REPORT CSECT TOTAL CHUNK 512 CHUNK 64
```

- XML document format (must be *SirTune* 7.2 or later)

  Report parameters are case-sensitive XML elements or attributes contained in a `SirtuneInput` document node.

  Most of the parameters represented as single control-card statements (as described above) are available in XML, but they are represented in lowercase or mixed-case, and some are individual elements while others are attributes of elements. The details about the coding location of each parameter are in "Using XML input for report configuration" on page 53.

  Unlike the control-card input format, where multiple variations on a report can be specified on a single line, in XML format each `<report>` element specifies exactly one report.

Here is an XML version of the control-card example above:

```
<SirtuneInput maxDelay="10" top="100" title="ULSPFPRO">
  <range start="9:30" end="12:30"/>
  <resolution value="50"/>
  <reportFormat>
    <title>Sirtune Reports for ULSPFPRO</title>
    <charactersPerLine>78</charactersPerLine>
    <linesPerPage>55</linesPerPage>
    <tableOfContents>top</tableOfContents>
  </reportFormat>
  <report type="cfrroot"/>
  <report type="what" state="crfbany"/>
  <report type="quad" state="rung"/>
  <report type="evali" state="rung"/>
  <report type="eval" state="rung"/>
  <report type="eval" state="rung" chunk="32"/>
  <report type="eval" state="rung" chunk="4">
  <report type="eval" state="rung" chunk="400">
  <report type="eval" state="rung" chunk="4000">
  <report type="csect" chunk="total"/>
  <report type="csect" chunk="512"/>
  <report type="csect" chunk="64"/>
</SirtuneInput>
```

7.1 Configuration parameters

The following subsections describe the individual parameters you can use to configure a *SirTune* report. Most of the parameter names and descriptions refer to the control-card statement format that was the only control-card option prior to *SirTune* version 7.2, although pertinent XML format information is also provided. For more information about using XML for the control-card input, see “Using XML input for report configuration” on page 53.

Where applicable, the titles of the following subsections also show how to specify the parameter in control-card statement form:

- The minimum allowable abbreviation for each statement is indicated with uppercase characters, while optional characters are indicated with lowercase.

  Names are case-sensitive and these name abbreviations are not permitted in XML-formatted input.

- Optional statement keywords are enclosed in brackets ([]), and alternatives are separated with a vertical line (|).

7.1.1 CHARACTERSPERLINE or CPL

This statement determines the width of an output report line. For example, either of the following statements sets the report output width to 80 characters:

```
CHARACTERSPERLINE 80

CPL 80
```

Valid line-width values are integers between 78 and 256. If a value outside this range is specified, CPL is set to 132.

If you are using XML input to TUNERPTI, the `<cpl>` or `<charactersPerLine>` element specifies the line width. For more information about this, see "<reportFormat>: Controlling report format" on page 56.

7.1.2 comp31

Status: This parameter is new for Version 7.2. It can only be specified when using XML input to TUNERPTI, and it is only valid for *Model 204* V6R3 and later.

comp31 indicates the maximum size of compilation table for a procedure that will be stored in 31-bit storage. Above this size, compilations are moved to 64-bit storage. The default value is x'1FFFF', or 131,071.

comp31 is specified as an attribute on the `SirtuneInput` element (see "Using XML input for report configuration" on page 53); for example:

```
<SirtuneInput maxDelay="10" comp31="200000">
```

The size of a saved compilation table for a procedure is the size of the procedure in QTBL bytes divided by the minimum QTBL chunk size, times 20:

```
(QTBLsize/QTBLchunk)*20
```

If a procedure's QTBL is 20,000 bytes and the smallest chunk size requested in a report is 200, this procedure will require (20,000/200)*20, or 2,000 bytes. More common are procedure sizes of around 800,000 bytes, with a chunk size of 4, theoretically requiring (800,000/4)*20, or 4,000,000 bytes. This is actually an upper bound, as you never get more than one chunk per line, and few lines of code generate only 4 bytes of QTBL. More typical is about 40 bytes per chunk (even with a chunk size of 4), which produces a compiled table of 200,000.

There is no reason to set comp31 unless your site is running out of 31-bit storage under *Model 204* V6R3 or later. Prior to V6R3, *Model 204* did not support 64-bit storage, so comp31 has no effect.

Note: As described in the ***Sirius Mods Command and Parameter Reference Manual***, the CURREP31, CURREP64, HGHREP31, and HGHREP64 system parameters indicate the current and greatest amounts of 31-bit and 64-bit virtual storage used for a report. The MAXREP31 and MAXREP64 parameters indicate the maximum amount allowed.

7.1.3 DATaset list

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement indicates which sample datasets that make up SIRTUNED are to be included in the report. A separate `INFO` report is produced for each dataset selected, and all samples from the selected datasets are combined to produce composite results for other reports. Datasets are processed in order: if a dataset is selected and causes some TUNR error message, reporting ends without proceeding to other datasets.

If the DATASET statement is not specified, only the first sample dataset is included in the report. This is true even if multiple sample datasets were physically concatenated into a single dataset with IEBGENER or COPYFILE before being passed to the reporting module.

The *list* value is a list of dataset numbers or ranges of numbers. For example, the following indicates that datasets 1 through 4 are to be used:

```
DATASET 1-4
```

The following indicates that datasets 2, 4, and 5 through 6 are to be used:

```
DATASET 2 5-6 4
```

More than one DATASET statement can appear in SIRTUNEI, and their effect is cumulative, so

```
DATASET 2
DATASET 4-6
```

is the same as

```
DATASET 2 4-6
```

If an asterisk (*) character appears after a DATASET statement, all datasets are used in the report. Thus the following

```
DATASET 1 3-5 8 *
```

is the same as

```
DATASET *
```

since in both cases, all datasets are used in the report.

There is no harm in specifying dataset numbers greater than the number of sample datasets in SIRTUNED, though of course, these dataset numbers will not be included in the report.

RANGE, SKIP, and FOR statements operate individually on each component sample dataset in a concatenation. For example, suppose SIRTUNED consists of three concatenated sample datasets collected over the following dates and times:

Dataset 1 January 5 at 5:30 AM through January 8 at 11:00 PM

Dataset 2 January 10 at 1:00 PM through January 15 at 10:00 PM

Dataset 3 January 11 at 5:00 AM through January 14 at 10:00 PM

In this example, datasets 2 and 3 are associated with two different Onlines that run concurrently. The SIRTUNEI statements

```
RANGE 11:00 12:00 SKIP 3 FOR 2
DATASET 1-3
```

would result in the range of 11:00 AM to 12:00 noon being used on the following days for each dataset:

Dataset 1 January 8
Dataset 2 January 14-15
Dataset 3 January 14-15

To determine the time ranges and characteristics of each individual dataset in a concatenation, simply run a report against the concatenated SIRTUNED with the following commands in SIRTUNEI:

```
REPORT NODEFAULT
REPORT INFO
```

It should be possible to produce most *SirTune* reports with any concatenation of datasets. There are certain reports, however, that require all sample datasets to have been produced with some consistency among the Onlines. These reports are:

CSECT? The *Model 204* load modules must be identical among runs (except for ZAPs or Early Warnings).

STATE ???? QUAD The *Model 204* load modules must be from the same release and must have run with identical $function tables.

SERVIO The OnlineS must have run with identical numbers of server datasets (CCASERVR, CCASERV1, etc.).

STATE RUNGM/RUNGS The OnlineS must have all run with NMPSUBS greater than 0 (the MP feature is turned on).

Even though all other reports are allowed for concatenated sample datasets, use caution when interpreting the results. It is up to you interpret how changes between runs could affect the results. These changes might include:

- User Language code changes
- Changes in *Model 204* parameters
- Changes in the *Model 204* load module
- Changes in *SirTune* parameters
- Number of samples collected in each sample dataset

In general, it is unlikely that concatenating datasets collected in different environments or many months apart will produce meaningful reports.

Suppose, under MVS, the SIRTUNER or SIRTUNEREPORT job logically concatenates five sample datasets associated with the 5 weekdays as follows:

```
//SIRTUNED DD  DSN=SIRTUNE.SAMPLE.MON,DISP=SHR
//         DD  DSN=SIRTUNE.SAMPLE.TUE,DISP=SHR
//         DD  DSN=SIRTUNE.SAMPLE.WED,DISP=SHR
//         DD  DSN=SIRTUNE.SAMPLE.THU,DISP=SHR
//         DD  DSN=SIRTUNE.SAMPLE.FRI,DISP=SHR
```

To produce a report for Tuesday and Wednesday simply code the following input statements:

```
DATASET 2 3
```

7.1.4 LINESPERPAGE or LPP

This statement determines the page length for report output. Any integer value is valid.

A header is printed at the top of each page. If 0, the default, is specified, no paging is performed, and headers are output at the beginning of each report.

For example, either of the following statements sets report page length to 55 lines:

```
LINESPERPAGE 55

LPP 55
```

When 55 lines have been printed, SIRTUNEREPORT prints a new header and resets the line counter to 0.

If you are using XML input to TUNERPTI, the `<lpp>` or `<linesPerPage>` element specifies the report page length. For more information about this, see “<reportFormat>: Controlling report format” on page 56.

7.1.5 MAPcore

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

When used in a CSECT breakdown report, this statement allows analysis of instructions outside the loaded *Model 204* module.

7.1.6 MAXDelay max_msec

This statement makes it possible to change the criterion used by the reporting module to determine if a sample should be discarded.

When the delay between the time a sample was supposed to be collected and the time it actually was collected is greater than *max_msec* milliseconds, the sample is assumed to be biased and hence discarded. The default for *max_msec* is 10, which means that any sample that was delayed by more than 10 milliseconds from its intended time will be discarded.

If it is felt that reports are biased, with I/O producing instructions appearing unbelievably frequently in the RUNG state (obviously, this is very subjective), it might be worth reducing MAXDELAY to 1 with the following statement:

```
MAXDELAY 1
```

If there was indeed a bias, one would expect after this change to see more samples discarded and the percentages for the RUNG state to change for certain procedures and chunks of procedures.

If, on the other hand, *SirTune* was running in an environment where the Online was getting relatively slow service from the operating system, almost all samples might be discarded by the reporting module. In this situation, it might be worth increasing MAXDELAY to accept more samples, with the understanding that this might introduce biases in the reported data.

To increase MAXDELAY to 1000 milliseconds (1 second), enter this statement:

```
MAXDELAY 1000
```

Generally, setting MAXDELAY greater than 100 is likely to produce severe biases in Sirtune reports.

7.1.7 MIXed

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement indicates that the reporting module should issue all reports in mixed case. The default is to issue messages in mixed case except on Japanese operating systems. On these systems, the default is to issue all message in uppercase only. This option should be used on Japanese operating systems when mixed case messages are desired.

7.1.8 MPVirt

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement has meaning only if generating a report for data collected in an Online running MP/204. It indicates that a running user is considered to be running in parallel if it is in virtual or logical parallel mode, whether or not it is actually running in an offload subtask. By default, the *SirTune* reports only consider a user to be running in parallel if it is actually running on an offload subtask.

The MPVIRT statement is useful for estimating the “offloadability” of User Language code in systems that are not busy enough to force all offloadable code into a subtask.

Note: MPVIRT only has an effect on reports that are specific to states RUNGM or RUNGS.

It is not possible to split the reports in a run of *SirTune* so that some are generated with MPVIRT set and others are not. If MPVIRT appears anywhere in the input stream, all reports from that reporting run are generated using the MPVIRT option, whether the REPORT statements appear before or after the MPVIRT statement.

7.1.9 NOSeq

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement indicates that reporting module should not consider characters in columns 73 through 80 of SIRTUNEI as part of its input. If the input file has RECFM=F and LRECL=80, the reporting module ordinarily assumes that characters in columns 73 through 80 are sequence numbers and ignores them. If the input stream has any other format, the reporting module considers all characters in the input record as part of the control statements.

7.1.10 RANge start_time end_time [FOR for_num] [SKIP skip_num]

This statement makes it possible to limit the reports to a subset of the collected sample data. This is useful for solving performance problems that have different characteristics during different times of the day.

start_time and *end_time* can have the format HH, HH:MM or HH:MM:SS, where HH represents an hour of the day, MM represents minutes, and SS represents seconds. The maximum allowable time is 24 or 24:00 or 24:00:00, which corresponds to midnight. For example, the following statement indicates that only sample data collected in the first 1 PM to 3 PM range in the run should be used to generate the reports:

```
RANGE 13 15
```

When using XML format, RANGE is specified in lowercase as a child element of the document element, with "start", "end", "skip", and "for" as optional attributes:

```
<SirtuneInput top="50">
  <range start="13" end="15">
```

If the run came up in the middle of the specified range, the time from the start of the run to the end of the range is considered the first range. In the preceding example, if the run came up at 2:30 PM, the first range would extend from 2:30 PM to 3 PM. The following example indicates that only sample data collected in the first 11:30 AM to 1:30 PM range in the run should be used to generate the reports:

```
RANGE 11:30 13:30
```

This example indicates that only sample data collected in the first 30 seconds before 12 noon through 50 seconds after 1 PM range in the run should be used to generate the reports:

```
RANGE 11:59:30 13:00:50
```

If *start_time* is greater than *end_time* the times are assumed to wrap. For example. the following statement indicates that only sample data collected in the first 11:00 PM to 1:00 AM range in the run should be used to generate the reports:

```
RANGE 23:00 1:00
```

If sample data for a run was collected over more than 24 hours, a range might occur more than once in the sample dataset. Ordinarily, only the first range is included in the reports. To include a specific number of occurrences of the indicated range, the FOR clause should be included on the RANGE statement, followed by *for_num*, where *for_num* can be either a whole number or an asterisk (*). A whole number indicates the number of occurrences of the range are to be included in the report. An asterisk indicates that all occurrences of the range are to be included in the report.

For example, the following statement indicates that the first 5 9:30 AM to 11:30 AM ranges are to be included in the reports:

```
RANGE 9:30 11:30 FOR 5
```

The data in these ranges are combined to produce a single set of reports. The following statement indicates that all 10 AM to 3 PM ranges are to be included in the report:

```
RANGE 10 15 FOR *
```

To exclude the first occurrence or occurrences of the indicated range from the reports, include the SKIP clause on the RANGE statement, followed by *skip_num* (where *skip_num* must be a non-negative integer indicating the number of occurrences of the indicated range to skip). For example, to indicate that the second 10 AM to 12 noon range is to be included in the reports:

```
RANGE 10 12 SKIP 1
```

To indicate that the third, fourth, and fifth 1:30 PM to 3:30 PM ranges are to be included in the reports:

```
RANGE 13:30 15:30 SKIP 2 FOR 3
```

While multiple RANGE statements may be specified only the last one is used when specified as control cards. Conversely, only the first one is used when specified in XML input Thus,

```
RANGE 9:30 11:30
RANGE 13:30 15:30
```

is functionally equivalent to

```
RANGE 13:30 15:30
```

The position of RANGE statements relative to other statements is irrelevant.

7.1.11 REPort report_desc | NODEFAULT

This statement either requests a specified report or reports, or it indicates that the default reports are not to be generated (REPORT NODEFAULT). The actual reports that can be requested are listed in “SirTune Reports” on page 59.

If REPORT NODEFAULT is specified, it must be the first REPORT statement in the input stream (if input is XML formatted, the position of REPORT NODEFAULT is not critical).

The default reports produced by SIRTUNEI are the ones that are produced by the following statements:

```
REPORT INFO
REPORT SUMMARY
REPORT STATE RUNG WHAT EVAL CHUNK 4000
REPORT STATE RUNG EVAL CHUNK 400 CHUNK 4
```

In XML format, these default reports are specified like this:

```
<SirtuneInput maxDelay="10" top="50">
   <report type="info">
   <report type="state">
   <report type="waittype">
   <report type="what" state="rung">
   <report type="eval" state="rung">
   <report type="eval" state="rung" chunk="4000">
   <report type="eval" state="rung" chunk="400">
   <report type="eval" state="rung" chunk="4">
</SirtuneInput>
```

For more information about XML formatting for an individual report, see “<report>: Specifying individual reports” on page 57.

7.1.12 RESolution res_num [PROC pname] [FILE fname] [SUBSYS sname]

This statement makes it possible to reduce the reporting module's virtual storage utilization at the cost of reducing the resolution possible in producing procedure chunk reports. This statement should be of interest only to users for whom virtual and real storage use is a major concern.

Note: As described in the ***Sirius Mods Command and Parameter Reference Manual***, the CURREP31, CURREP64, HGHREP31, and HGHREP64 system parameters indicate the current and greatest amounts of 31-bit and 64-bit virtual storage used for a report. The MAXREP31 and MAXREP64 parameters indicate the maximum amount allowed.

Storage use by the reporting module is directly related to the total number of chunks for all procedures for which compilation information is saved. The number of chunks required for each such procedure is inversely related to its minimum resolution. Thus by increasing the minimum resolution for a procedure, it is possible to decrease the amount of virtual storage used for that procedure. The default resolution for all procedures is 4.

The *res_num* value specifies the minimum chunk resolution in QTBL bytes for all or the indicated procedures. For a discussion of chunks and resolution, see “REPORT STATE state_name activity” on page 67.

If not followed by any other keywords, this statement sets the default minimum resolution for all procedures. For example, this sets the default minimum resolution for all procedures to 100:

```
RESOLUTION 100
```

There can be an unlimited number of this kind of RESOLUTION statements, but only the last one is used to set the default minimum resolution when the input stream is formatted as control cards. When XML input is used, only the first RESOLUTION statement is used. Thus,

```
RESOLUTION 50
RESOLUTION 100
RESOLUTION 150
```

is functionally equivalent to

```
RESOLUTION 150
```

Likewise, this XML input

```
<resolution value="150">
<resolution value="100">
<resolution value="50">
```

is functionally equivalent to

```
<resolution value="150">
```

res_num may optionally be followed by one or more keywords (PROC, FILE, or SUBSYS), with each keyword followed by a resolution that applies only to procedures that meet the keyword's criterion. For example, this statement indicates that a resolution of 1000 should be used for all procedures in subsystem HOHO:

```
RESOLUTION 1000 SUBSYS HOHO
```

Using XML input, the keywords are the same except in lowercase:

```
<resolution value="1000" subsys="HOHO">
```

This statement indicates that a resolution of 500 should be used for all procedures in procedure file PROCA:

```
RESOLUTION 500 FILE PROCA
```

Or, in XML mode:

```
<resolution value="500" file="PROCA">
```

The following indicates that a resolution of 700 should be used for the procedure named PRE-PROC.A:

```
RESOLUTION 700 PROC PRE-PROC.A
```

Wildcards can also be used. To indicate that a resolution of 100 should be used for any procedure whose name begins with PRE-, you can specify:

```
RESOLUTION 100 PROC PRE-*
```

The following statement indicates that a resolution of 300 should be used for any procedure in a subsystem whose name ends with XREF:

```
RESOLUTION 300 SUBSYS *XREF
```

For more details on the use of wildcards, see "Wildcard Strings in SirTune and SIRTUNER Statements" on page 95.

More than one condition can be specified on a single RESOLUTION statement, in which case the resolution specified applies only to procedures that match all the keyword criteria. For example, the following statement sets a resolution of 1000 for any procedure whose name ends in TEST *and* is in a procedure file whose name begins with SIR:

```
RESOLUTION 1000 FILE SIR* PROC *TEST
```

In XML mode, the previous statement is:

```
<resolution value="1000" file="SIR*" proc="*TEST">
```

An unlimited number of qualified RESOLUTION statements may appear in the report configuration input. If more than one does occur in the input stream, each procedure for which compilation data is encountered is compared against each RESOLUTION statement, in the order in which they appear. As soon as a match is found, the resolution is set for the procedure, and no more RESOLUTION statements are scanned.

Thus, if SIRTUNEI contains

```
RESOLUTION 1000 SUBSYS SIR*
RESOLUTION 500 FILE TEST*
RESOLUTION 200 PROC *TEMP*
```

any procedure in a subsystem whose name begins with SIR would have a resolution of 1000, even if it ran out of a procedure file whose name begins with TEST, or if the procedure name contains the characters T E M P.

The resolution used for specific procedures with the resolution statements in the preceding example are listed below:

Subsystem	Procfile	Procedure name	Resolution
TOOLSYS	GENFILE	BOP.PROD.L	default
SIRJUNK	JUNKPROC	P.JUNK.PROC	1000
BIGSYS	TESTFILE	P.BARTLIST	500
BIGSYS	PROCX	EXPER.TEMP.P	200
SIRJUNK	TESTFILE	P.BARTLIST	1000
SIRJUNK	PROCX	EXPER.TEMP.P	1000
BIGSYS	TESTFILE	EXPER.TEMP.P	500
SIRJUNK	TESTFILE	EXPER.TEMP.P	1000

The suggested use of the RESOLUTION command is that the default is left at 4 and that resolution is set to a high value for procedures that are known to be compiled but are not likely to show much activity. If one or more of these procedures shows up on a CHUNK report at a resolution smaller than that set by the RESOLUTION commands for the procedures, do either of the following:

- Modify the RESOLUTION command.
- Add extra RESOLUTION commands to explicitly set the resolution at a lower value for the procedures that appear on the CHUNK report.

For example, suppose the following statement is specified, and a procedure named PRE.JUNK.PROC in subsystem TEST appears on a report for chunk size 100:

```
RESOLUTION 400 SUBSYS TEST
```

Since the minimum chunk size that SIRTUNER or SIRTUNEREPORT can use for procedures in subsystem TEST is 400 (because of the RESOLUTION statement), the report with chunk size 100 is comparing chunks of size approximately 100 with a chunk of size 400 in PRE.JUNK.PROC. Thus, a much larger segment of code is being totalled for PRE.JUNK.PROC than other procedures on this report, making this procedure look more important than it really is.

To correct this situation, you can either remove the RESOLUTION statement from SIRTUNEI or TUNERPTI and lose its virtual storage savings benefits, or you can add another RESOLUTION statement to override the subsystem wide RESOLUTION statement for the specific procedure in question. The following sequence would accomplish this.

```
RESOLUTION 4 SUBSYS TEST PROC PRE.JUNK.PROC
RESOLUTION 400 SUBSYS TEST
```

7.1.13 TITle title_string

This command makes it possible to specify a title to appear on the top line of each page of the *SirTune* report. The default for *title_string* is `SIRTUNE`.

TITLE makes it easy to distinguish the output from multiple SIRTUNER reports. The title string can be made up of an arbitrary number of characters, including blanks. Only the first 62 characters of *title_string* are used to generate the title.

An example of a valid TITLE statement follows:

```
TITLE Production run after first round of changes
```

When using XML input in version 7.2 and later, you can specify the title as either a child element of the document node, like this:

```
<SirtuneInput>
  <title>Sirtune June 16, 2010 for THX1138</title>
  ...
```

Or you can specify it as a child element of the reportFormat node, like this:

```
<SirtuneInput>
  <reportFormat>
    <title>Sirtune June 16, 2010 for THX1138</title>
  </reportFormat
  ...
```

If specified in both places, the child element of reportFormat will override the child element of the document node. The reporting module only pays attention to the first title specified in either location in the input document.

7.1.14 TOP num_top

This statement specifies the number of top entities to be displayed on any report that ranks multiple entities. The default for *num_top* is 50. This default can be overridden by a TOP keyword on a REPORT statement (“The TOP parameter” on page 73).

For example, if you want to report on the top 100 procedures or chunks of procedures on all reports, place this statement in the input stream:

```
TOP 100
```

Or, with XML input in 7.2 and later, enter `top` as an attribute on the document node:

```
<SirtuneInput top="100">
```

Generally, it is not profitable to look at anything past the top 50 entities in any report, since less than 2% of resource usage can be attributed to any entity not in the top 50. It is generally not worth optimizing something that accounts for so little resource usage.

7.1.15 TABLEOFCONTENTS or TOC

This statement determines whether a table of contents is produced and where it is placed in the report. Although TWOPASS can still be used to specify the table of contents placement, the TABLEOFCONTENTS statement provides more control.

Valid values for TABLEOFCONTENTS are `TOP`, `BOTTOM` and `NONE`. By default, the table of contents is placed at the end of the report.

If you are using XML input to TUNERPTI, the `<toc>` or `<tableOfContents>` element specifies characteristics of the table of contents. For more information, see "<reportFormat>: Controlling report format" on page 56.

7.1.16 TWOpass

This statement indicates that the reporting module should place the table of contents for the output report at the start of the report rather than at the end. By default, the table of contents is placed at the end of the report.

As of version 7.2, this parameter is misnamed: the reporting module no longer needs to make two passes of the data, regardless of where the table of contents is positioned.

If you are using XML input to TUNERPTI, the `<toc>` or `<tableOfContents>` element specifies characteristics of the table of contents. See "<reportFormat>: Controlling report format" on page 56.

7.1.17 UPper

Status: This parameter is deprecated as of Version 7.2. If it is specified as an input parameter, it is ignored.

This statement indicates that the reporting module should issue all messages in uppercase only. The default is to issue messages in mixed case, except on Japanese operating systems, where the default is to issue messages in uppercase only.

This option should be used on systems where mixed-case messages might not be displayed correctly on terminals or printers.

7.2 Using XML input for report configuration

As of version 7.2, it is recommended that the control parameters described in “Configuration parameters” on page 38 be formatted as an XML document in TUNERPTI. As shown in the example that follows, this XML document is characterized by:

- A root node, `<SirtuneInput>`, whose attributes are report parameters.
- Optional sub-elements (“child” elements) for other parameters, including `<reportFormat>`, which provides output formatting not available prior to version 7.2.
- A `<report>` element for each report, specified with appropriate attributes as described in “REPort report_desc | NODEFAULT” on page 46.

A sample TUNERPTI input document follows. The entire set of TUNERPTI XML document element and attribute options is shown in “A template for the XML input” on page 54.

```
<SirtuneInput maxDelay="10" top="100">
   <title>ULSPFPRO on Sirius</title>
   <reportFormat>
     <linesPerPage>55</linesPerPage>
     <charactersPerLine>78</charactersPerLine>
     <tableOfContents>top</tableOfContents>
   </reportFormat>
   <report type="info"/>
   <report type="waittype"/>
   <report type="state"/>
   <report type="sysparm"/>
   <report type="csect" top="100"/>
   <report type="csect" chunk="64" top="100"/>
   <report type="csect" chunk="4" top="100"/>
   <report type="csect" task="maintask" top="100"/>
   <report type="csect" chunk="64" task="maintask" top="100"/>
   <report type="eval" chunk="400" state="wchkpo"/>
   <report type="eval" chunk="4" state="wchkpo"/>
   <report type="eval" state="wbuff"/>
   <report type="eval" chunk="400" state="wbuff"/>
   <report type="eval" chunk="4" state="wbuff"/>
   <report type="whatc"/>
   <report type="whatc" chunk="64"/>
   <report type="servuse" chunk="10000"/>
   <report type="servio"/>
</SirtuneInput>
```

7.2.1 General notes on TUNERPTI formatting

The following notes further describe the format of a TUNERPTI XML document like the example above.

- As with all XML, the element and attribute names in a TUNERPTI XML document are case sensitive. “SirtuneInput” is **not** validly entered as “SirTuneInPut” or as any other combination of uppercase and lowercase letters except “SirtuneInput”. Element and attribute *values* are case-insensitive.

  A few of the parameter names are in mixed case, but for ease of coding, most elements and attributes are restricted to lowercase characters.

- While multiple reports can be specified on a single line in the older control card format, each `<report>` element in the XML input represents a single report.

- The configuration parameters `top`, `maxDelay`, and `comp31` are specified as attributes of the `<SirtuneInput>` document element.

  The configuration parameters `title`, `range`, and `resolution` are specified as sub-elements of `<SirtuneInput>`, and their values are element content.

  `title` can alternately be specified as a sub-element of `<reportFormat>`.

- Parameters affecting the report layout are specified as sub-elements of the `<reportFormat>` element.

- While the old-style card-image input format will continue to be supported after version 7.2, newer features are only available via XML input. Features such as user-specified characters-per-line and lines-per-page are already available only when using XML input to TUNERPTI.

7.2.2 A template for the XML input

This is a fully-qualified template for the TUNERPTI input XML document. It shows the document format and its possible XML elements, attributes, and content.

Square brackets (`[ ]`) enclose values you supply. The type, format, or explicit set of valid options for these values is indicated within the brackets. “Guidelines for the three main XML elements” on page 56 provides additional details.

```
<SirtuneInput top="[integer]"
              maxDelay="[integer]"
              comp31="[integer]">

  <range start="[HH|HH:MM|HH:MM:SS]"
           end="[HH|HH:MM|HH:MM:SS]"
           for="[integer]"
          skip="[integer]" />
```

```
  <resolution value="[integer]"
              subsys="[subsysname]"
                file="[filename]"
                proc="[procname]" />

  <title>[62-character title]</title>

  <reportFormat>
     <title>[62-character title]</title>
     <charactersPerLine>[integer]</charactersPerLine>
     <linesPerPage>[integer]</linesPerPage>
     <tableOfContents>[top|bottom|none]</tableOfContents>
   </reportFormat>

   <report type="[cfroot|csect|csectm|csects|diskio|info|
                  quad|quadcm|quadcs|servio|servuse|state|
                  summary|sysparm|whatc|whatcm|whatcs|
                  waittype|nodefault]"
           top="[integer]"
           chunk="[total|table|integer]"
           task="[subtask|maintask]"

<!--if a REPORT STATE, the following are the "type"-->
<!--and "state" attribute options-->
           type="[what|comp|load|eval|evali|quad|chunk|
                  ifjob|ifcomp|ifjcomp|iffunc|ifchunk]"
           <!--"state" options include primary states:->
           state="[blkin|blkiu|blkon|blkou|redy|rung|rungm|
                   rungs|swpgi|swpgobn|swpgobu|swpgow|wpst|
                   wtsv|
           <!--composite states:->
                   all|alli|alln|blk|blki|blkn|blko|blku|
                   oservn|oservu|oservw|redyr|runbl|swpg|
                   swpgo|swpgob|
           <!--wait types:->
                   wmisc|wdisk|wusero|wuseri|woperi|wdumpo|
                   wdumpi|wenque|wbuff|wpst|wifam|wsleep|
                   wjrnlo|wchkpo|wwrite|warbmo|wchkpr|wdisk|
                   wdead|wvsami|wlogin|wcfrex|wcfrsh|wvtbuf|
                   wconvi|wconvo|wsctyi|ws$wai|wn$wai|wuldb2|
                   wcfr|wlog|
           <!--critical file resources:->
                   cfrhany|cfrhdir|cfrhind|cfrhexs|cfrhrec|
                   cfrbany|cfrbdir|cfrbind|cfrbexs|cfrbrec]">
</SirtuneInput>
```

7.2.3 Guidelines for the three main XML elements

This section shows how to assemble the principal elements of the XML format supported as control input to *SirTune* Version 7.2 and later.

7.2.3.1 <SirtuneInput>: Document root element

The root element (which is the outermost element) of the input XML document is `<SirtuneInput>`. The optional report parameters `maxDelay`, `top`, and `comp31` are entered as attributes of this element:

```
<SirtuneInput maxDelay="10" top="100" comp31="200000">
```

7.2.3.2 <reportFormat>: Controlling report format

This element is optional; if specified, it must be a child of the document element. `<reportFormat>` contains child elements that control the formatting of report output. Specifically, you can define the number of lines per page and characters per line, and the position of the table of contents. For example:

```
<reportFormat>
  <linesPerPage>0</linesPerPage>
  <charactersPerLine>72</charactersPerLine>
  <tableOfContents>Bottom</tableOfContents>
</reportFormat>
```

- `<linesPerPage>` accepts integer values, 0 or greater. When the reporting module reaches linesPerPage output lines, a new header is printed and a new page is begun. A 0 setting turns off paging; in that case, a header line is produced at the beginning of the report only.

- `<charactersPerLine>` accepts integer values 40 or larger. A non-numeric value or a value lower than 40 is converted to 40. The default value is 78.

  If the report needs to print a line longer than charactersPerLine, the line wraps at charactersPerLine+1.

- `<tableOfContents>` (or its synonym `<toc>`) can be set to `Top`, `Bottom`, or `None`, specifying the location of the table of contents of the *SirTune* report. The default location is the bottom of the report.

7.2.3.3 <report>: Specifying individual reports

Each report is specified as a single, self-terminating, child element of <SirtuneInput>. Report details are specified as attributes of the <report> element:

```
<SirtuneInput>
   <report type="csect" chunk="64" task="maintask" top="100"/>
</SirtuneInput>
```

All <report> elements are self-terminating. All reports except STATE reports are formatted as in the above example: the report type is specified on the type attribute, and any sub-parameters are specified (in lowercase) as attributes of the <report> element. Report details and sub-parameters are described in "SirTune Reports" on page 59.

STATE reports ("REPORT STATE state_name activity" on page 67) are specified with an activity as the type attribute and the user state as the state attribute, as shown in the following example:

```
<SirtuneInput>
   <report type="quad" state="cfrbexs"/>
   <report type="eval" state="cfrbany"/>
   <report type="eval" chunk="4" state="rungm"/>
   <report type="evali" state="rung"/>
   <report type="evali" state="rungm"/>
   <report type="comp" state="rung"/>
</SirtuneInput>
```

Valid type activity values and state values are discussed or referred to in "REPORT STATE state_name activity" on page 67, and they are also summarized in "A template for the XML input" on page 54.

The order in which attributes are specified in the <report> element is not important.

CHAPTER 8 SirTune Reports

The *SirTune* reporting module produces a listing that is made up of one or more reports. These reports can be used to tune problem areas in an Online or BATCH204 job.

In *SirTune* versions prior to 7.2, reporting is done via SIRTUNER, an assembler module that is separately distributed and installed. In version 7.2 and later, reporting is done by SIRTUNEREPORT, a User Language program that is distributed in the *UL/SPF* file SIRIUS.

SIRTUNER cannot be used with *SirTune* 7.2 and later. Conversely, the SIRTUNEREPORT program cannot be used with versions prior to 7.2 of the *Sirius Mods*, since it is based on the `sirtuneReport` method of the Dataset class which was first implemented in version 7.2.

While it is possible to run SIRTUNEREPORT inside a *Model 204* Online, it is strongly recommended that it be run in a standalone batch job, as it can consume considerable resources from an Online job. If the SIRIUS file cannot be allocated to a batch job, SIRTUNEREPORT can be moved to another procedure file.

To run SIRTUNEREPORT from another file:

1. Move SIRTUNEREPORT and the included procedure SHARED_REPORT to the new file.

2. Change the single reference to SIRIUS to the new file name (this reference is an INCLUDE statement near the top of the procedure, after the variable declarations).

SIRTUNER accepts report specifications in control-card format on the SIRTUNEI input DD card. SIRTUNEREPORT accepts report specifications either in control-card format or in XML format on the TUNERPTI input DD card.

If not suppressed with the REPORT NODEFAULT statement (see “Report” on page 46), the reporting module always produces a default set of reports. Additional reports are requested with the REPORT statement.

The default reports produced are identical to the reports you get with the following REPORT statements:

```
REPORT INFO
REPORT SUMMARY
REPORT STATE RUNG WHAT EVAL CHUNK 4000
REPORT STATE RUNG EVAL CHUNK 400 CHUNK 4
```

In XML format, these default reports are specified like this:

```
<SirtuneInput maxDelay="10" top="50">
   <report type="info">
   <report type="state">
   <report type="waittype">
   <report type="what" state="rung">
   <report type="eval" state="rung">
   <report type="eval" state="rung" chunk="4000">
   <report type="eval" state="rung" chunk="400">
   <report type="eval" state="rung" chunk="4">
</SirtuneInput>
```

Note: The XML equivalent to the REPORT SUMMARY control-card statement is two elements:

```
<report type="state">
<report type="waittype">
```

In addition to these, a variety of other reports can be produced upon request. These reports can are requested via the REPORT statement (see example in “Generating reports prior to Sirius Mods 7.2: MVS” on page 33 or “Generating reports prior to Sirius Mods 7.2: CMS” on page 34). The reports appear in the same order in the output (TUNERPTO) as they appear in the input (TUNERPTI).

The main considerations to use in determining which reports to produce are (in descending order of importance):

- The completeness of the reports in pointing out problem areas.
- The ease with which the reports can be read.
- The CPU and storage costs of producing the reports.

Put another way, one should strive to get as much data as possible in SIRTUNER's listing without increasing the number of reports to such an extent that the listing becomes unwieldy and overwhelming to work with.

Sirtune reports are described in the following subsections. Optional parameters are listed in brackets (`[]`) and alternative parameters are separated by vertical bars (`|`).

Note: Due to format changes in the sample dataset, only the SIRTUNEREPORT program can process a *SirTune* version 7.2 and later sample dataset.

Note: Due to format changes in the sample dataset, only releases 1.6 and later of SIRTUNER can process a *SirTune* version 7.0 and later sample dataset.

Note: Due to format changes in the sample dataset in *SirTune* version 1.5, earlier releases of SIRTUNER cannot process a *SirTune* 1.5 or higher sample dataset.

8.1 REPORT CFRROOT

The CFRROOT report can be used to determine the root bottleneck behind critical file resource enqueuing. Since critical file resource enqueuing is never a bottleneck in itself but will exacerbate some other bottleneck, this report can be useful in determining what is actually behind most of the critical file resource enqueuing on a system.

Generally, one would expect disk I/O to be the root bottleneck behind critical file resource enqueuing. Unfortunately, this is also usually the most complex bottleneck to correct. If disk I/O shows up as the top root bottleneck on the CFRROOT report, the next step should probably be to look at the STATE CFRB??? reports to try to locate the programs or statements that produce the disk I/O that results in critical file resource enqueuing. It might also be useful to look at STATE WDISK reports and DISKIO reports to get information on the cause of all waits for disk I/O.

Other than disk I/O, the most likely root causes of critical file resource enqueuing are:

Journal I/O This cause might be reduced by placing the journal on a faster device (DASD fast write is especially helpful here) or by reducing the quantity of journal data being produced (a REPORT STATE WJRNLO might help isolate code that is flooding the journal).

Checkpoint I/O This cause might be reduced by placing the checkpoint dataset on a faster device (DASD fast write is especially helpful here).

Arbitration This shows up as ARBMO and is most likely caused by journal I/O or checkpoint I/O. A reasonable guess for which of these is the most likely culprit is provided by the positions of journal and checkpoint I/O in the CFRROOT report.

CPU A CPU bottleneck can exacerbate the effect of I/O waits on critical file resource enqueuing. CPU tuning, processor upgrades and the MP/204 feature are all options in correcting CPU bottlenecks.

SERVER A server bottleneck can exacerbate the effect of I/O waits and/or a CPU bottleneck on critical file resource enqueuing. In fact, a server bottleneck can interact with another I/O bottleneck and critical file resource enqueuing in a positive feedback loop that turns a minor bottleneck into a disastrous performance problem. While it is possible that increasing the number of servers might break this feedback loop it is more likely that attacking the I/O bottleneck will improve performance more, even if SERVER shows up at the top of the CFRROOT report.

If anything else appears as a significant root cause of critical file resource enqueuing, it is likely to indicate a severe but possibly easily correctable problem. Sirius Software support should be able to help diagnose and correct such a problem.

8.2 REPORT CSECT | CSECTM | CSECTS TOTAL | CHUNK ch_size

The CSECT/CSECTM/CSECTS reports break down *Model 204* CPU usage by CSECT. These reports are generally only of interest to *Model 204* internals experts, though they might be of interest in shops that have extensive in-house $function libraries.

The CSECT report shows CPU usage by CSECT. When running the MP/204 feature, the CSECTM report shows maintask CPU usage by CSECT, and the CSECTS report shows subtask CPU usage by CSECT. If not running the MP/204 feature or running the feature with 0 subtasks, CSECT is equivalent to CSECTM, and CSECTS is always 0.

Note: If using XML for the parameters ("Using XML input for report configuration" on page 53), a single convention can handle these different CSECT report variations: specify a "task=maintask" or "task=subtask" attribute for a "type=csect" report to invoke, repectively, a CSECTM or a CSECTS report. For example, to get a CSECTM report:

```
<report type="csect" task="maintask">
```

The CSECT|M|S reports are further qualified by a TOTAL or CHUNK keyword:

- TOTAL requests a breakdown of CPU usage by whole CSECTs. For example, to produce a breakdown of CPU usage where each entry in the report is an entire CSECT, you use:

  ```
  REPORT CSECT TOTAL
  ```

  Since TOTAL is the default, a `REPORT CSECT` statement is equivalent to `REPORT CSECT TOTAL`.

  If using XML in the report configuration input, `REPORT CSECT TOTAL` is equivalent to `<report type="csect" chunk="total"/>`.

- CHUNK request a breakdown of CPU usage by pieces of each CSECT. The size of each piece or chunk is *ch_size* bytes of object code. Thus to get a breakdown of CPU usage by 128-byte chunks of object code, code the following in SIRTUNEI:

  ```
  REPORT CSECT CHUNK 128
  ```

You can specify multiple breakdown types on a single REPORT CSECT statement. For example, this statement requests a breakdown of CPU usage by entire CSECTs, by chunks of 512 bytes, and by chunks of 64 bytes:

```
REPORT CSECT TOTAL CHUNK 512 CHUNK 64
```

Note: You can use the MAPCORE statement ("MAPcore" on page 43) in the report configuration input to specify that instructions outside of the loaded *Model 204* module are to be treated as a "CSECT," allowing analysis of CPU usage in the entire address space.

8.3 REPORT DISKIO TOTAL | TABLE | CHUNK ch_size

The DISKIO report provides a breakdown of waits on *Model 204* disk I/O (database I/O). The possible breakdowns available are:

TOTAL This breaks down waits on *Model 204* disk I/O by database file. This is useful for determining which files have the heaviest I/O activity.

TABLE This breaks down waits on *Model 204* disk I/O by tables within each database file. The five database file tables are the FCT, Table A, Table B, Table C, and Table D. This breakdown is useful for determining if it is worth tuning the use of specific tables in the database files.

CHUNK This breaks down waits on *Model 204* disk I/O by groups or chunks of a specific number (*ch_size*) of pages within each file table. This can be particularly useful in isolating device performance or contention problems that affect part of a table that resides on multiple disks. In addition, it can be useful in isolating “hot” areas of activity that might benefit from being placed on a cached or faster DASD.

This breakdown is generally only useful when using extremely large tables. Even in these cases, it is recommended that *ch_size* be set to a large value (>1000) to produce meaningful results.

To produce the DISKIO reports, COLLECT DISKIO must have been specified in the *SirTune* data collection facilities input stream (SIRTUNEI).

To produce a report breaking down waits on disk I/O by files, code the following in SIRTUNEI:

```
REPORT DISKIO TOTAL
```

To produce a report breaking down waits on disk I/O by tables within files, code the following in SIRTUNEI:

```
REPORT DISKIO TABLE
```

To produce a report breaking down waits on disk I/O by chunks of 10000 pages within each table within files, code the following in SIRTUNEI:

```
REPORT DISKIO CHUNK 10000
```

Multiple breakdown types can be specified on a single REPORT DISKIO statement. For example, the following SIRTUNEI specification requests reports breaking down waits on *Model 204* disk I/O by file, tables within file and chunks of 5000 pages within tables:

```
REPORT DISKIO TOTAL TABLE CHUNK 5000
```

8.4 REPORT INFO

The INFO report provides environmental information about the *Model 204* ONLINE or BATCH204 program and *SirTune* data collection settings used in producing the sample dataset. This report includes settings of key *Model 204* parameters (NUSERS, NSERVS, NJBUFF, etc.) and *SirTune* parameters (sampling interval, COLLECT settings, etc.).

This is one of the default reports produced by SIRTUNER. It is recommended that this report always be included in SIRTUNER output, because it establishes the context of all other reports.

8.5 REPORT QUADC | QUADCM | QUADCS TOTAL | CHUNK ch_size

The QUADC/QUADCM/QUADCS reports break down *Model 204* CPU usage by quad and CSECT. These reports are generally only of interest to *Model 204* internals experts, though they might be of interest in shops that have extensive in-house $function libraries.

The QUADC report breaks down all CPU usage by quad and the CSECTs. Since the breakdown is a CSECT-level breakdown, the MPVIRT configuration parameter (“MPVirt” on page 44) has no effect on the QUADCM and QUADCS reports — only the real task on which a request was running is used to distinguish CSECTS and CSECTM.

When running the MP/204 feature, the QUADCM report shows maintask CPU usage by quad and CSECT, and the QUADCS report shows subtask CPU usage by quad and CSECT. If not running the MP/204 feature or running the feature with 0 subtasks, QUADC is equivalent to QUADCM, and QUADCS is always 0.

Note: If using XML for the parameters (“Using XML input for report configuration” on page 53), a single convention can handle these different QUADC report variations: specify a “task=maintask” or “task=subtask” attribute for a “type=quadc” report to invoke, repectively, a QUADCM or a QUADCS report. For example, to get a QUADCM report:

```
<report type="quadc" task="maintask">
```

These quad reports are further qualified by a TOTAL or CHUNK keyword:

- TOTAL requests a breakdown of CPU usage by quads and entire CSECT. For example, to produce a breakdown of CPU usage where each entry in the report is an entire CSECT/quad combination, you use:

  ```
  REPORT QUADC TOTAL
  ```

Since TOTAL is the default, a `REPORT QUADC` statement is equivalent to `REPORT QUADC TOTAL`.

If using XML in the report configuration input, `REPORT QUADC TOTAL` is equivalent to `<report type="quadc" chunk="total"/>`.

- CHUNK requests a breakdown of CPU usage by pieces of each CSECT for each quad. The size of each piece or chunk is *ch_size* bytes of object code. Thus to get a breakdown of CPU usage by 128-byte chunks of object code, code the following in SIRTUNEI:

```
REPORT QUADC CHUNK 128
```

You can specify multiple breakdown types on a single REPORT QUADC statement. For example, this statement requests a breakdown of CPU usage by entire CSECT/quad combination, and by CSECT chunks of 512 bytes, and by chunks of 64 bytes:

```
REPORT QUADC TOTAL CHUNK 512 CHUNK 64
```

Note: The MAPCORE statement ("MAPcore" on page 43) in the report configuration input specifies that instructions outside of the loaded *Model 204* module should be treated as a "CSECT," allowing analysis of CPU usage in the entire address space.

8.6 REPORT REPSTAT [RESET]

The REPSTAT report provides information on the performance of the report generator itself. The REPSTAT report breaks down the time for the report generation process into its individual components, CPU time, waiting for input buffer time, waiting for report buffer time, etc. In addition, the REPSTAT report provides information on above-the-line (31 bit) and below-the-line (24 bit) storage usage.

Note: This report is only of interest if the performance or storage usage of SIRTUNER is a concern. It is not supported in *SirTune* 7.2 and later (using the SIRTUNEREPORT User Language program), and a `repstat` keyword is *not* supported as a report type option in the XML format for specifying report configuration parameters ("A template for the XML input" on page 54).

The RESET parameter requests that usage counters be reset after the REPSTAT report. This makes it easy to determine the cost of actually producing a particular report. For example, in the following sequence the counters reported in the second REPSTAT report will show the cost of producing the REPORT STATE RUNG CHUNK 100 report:

```
REPORT REPSTAT RESET
REPORT STATE RUNG CHUNK 100
REPORT REPSTAT
```

Note that this report will not indicate the cost of actually collecting the data required by this report. This latter cost will often be significantly greater than the cost of actually generating the report.

When the REPSTAT report is used with the TWOPASS statement, the REPSTAT report will show results only in the second report-generation pass.

8.7 REPORT SERVIO

The SERVIO report provides a breakdown of waits on *Model 204* server I/O by server datasets (CCASERVR, CCASERV1, etc.). This is useful in identifying server datasets on disks that are not performing well because of contention problems, hardware problems, or server datasets with an unusually low level of activity, perhaps because of underallocated extents.

In general, ideal performance is achieved when all *Model 204* server datasets show the same level of activity and response. *Model 204*'s round robin, user-to-server, dataset allocation scheme ensures this on a gross level. Disparities in number of users waiting on specific server datasets could be indicative of a problem.

8.8 REPORT SERVUSE [CHUNK ch_size]

The SERVUSE report provides a breakdown of users in particular states by server size. This can be useful in determining optimal server size allocations. The states for which information is provided is a subset of all states described in "Model 204 States" on page 75.

Server sizes are grouped by chunk size (*ch_size*). The default for *ch_size* is 10000. This default results in 100 possible categories of server sizes between 0 and 1,000,000 bytes.

Groups with no observations are not displayed on the SERVUSE report.

The breakdowns are provided by the following states:

- REDYR

  Running or ready users. If state REDY was not collected, this is displayed as RUNG, meaning running users.

- SWPGI

  Users swapping into a server. If this state was not collected, it is not displayed on the SERVUSE report.

- OSERVW

  Users waiting on server or swapping out to wait for server. If neither state SWPGOW or WTSV was collected, this state is not listed on the SERVUSE report. If only one of these states was collected, the header appears as the collected state rather than “OSERVW”.

  This state is of particular interest in this report because it indicates the size of server that users required but were not able to get immediately. Thus, a large average for number-of-users in state OSERVW for server sizes 190000-199999 would indicate a requirement for more 200K servers.

- BLKIN

  Users blocked in server waiting on something other than user input. If this state was not collected, it is not displayed on the SERVUSE report.

- BLKIU

  Users blocked in server waiting on user input. If this state was not collected, it is not displayed on the SERVUSE report.

- OSERVN

  Users blocked out of server or swapping out to wait for something other than user input. If neither state BLKON or SWPGOBN was collected, this state is not listed on the SERVUSE report. If only one of these states was collected, the header appears as the collected state rather than “OSERVN”.

- OSERVU

  Users blocked out of server or swapping out to wait for user input. If neither state BLKOU or SWPGOBU was collected, this state is not listed on the SERVUSE report. If only one of these states was collected, the header appears as the collected state rather than “OSERVU”.

8.9 REPORT STATE state_name activity

The STATE report sorts users in a specified state (*state_name*) by their current processing activity (*activity*).

state_name can be any valid *SirTune* state, and the states are described in “Model 204 States” on page 75.

The activities are described below:

WHAT By the generic activity groupings used by *Model 204* in its "WHAT" flag settings. The most common of these activities is compiling, loading, and evaluating User Language procedures.

COMP By the name of the procedure(s) being compiled. Generally, this report is not of interest unless the WHAT report indicates a significant amount of resource usage in compilation.

LOAD By the name of the pre-compiled APSY procedure(s) being loaded. Generally, this report is not of interest unless the WHAT report indicates a significant amount of resource usage in loading.

EVAL By the name of the procedure(s) being evaluated. This report is useful in determining:

- Which procedures are worth looking at in more detail
- Whether any non-precompiled or non-APSY procedures are significant resource consumers.

This is important because, unless the ALLCOMP option ("ALLComp" on page 13) was specified for the data collection portion of *SirTune*, non-precompiled or non-APSY procedures will not show up on chunk reports.

EVALI By the name of the innermost procedure(s) being evaluated. This report is useful in environments where the same procedure(s) are INCLUDE'd in in several different pre-compiled APSY procedures. By getting a state breakdown by EVALI, all resource usage that occurs while running an INCLUDE'd procedure gets assigned to the INCLUDE'd procedure. This makes it possible to determine the possible gains from tuning such a procedure.

QUAD By the current *Model 204* quad being evaluated. Quads are the internal representation of User Language. Every User Language statement maps to one or more quads. Each $function is considered a separate quad in this breakdown.

While a breakdown by quad might generally only be of interest to a *Model 204* internals expert, these breakdowns might provide some hints about overall application characteristics, even to programmers unfamiliar with *Model 204* internals. The $function breakdown could be of particular interest in shops that have many and/or complex in-house $functions.

CHUNK ch_size By pieces of evaluating procedures. Every User Language procedure can be broken up into an arbitrary number of pieces. *SirTune* allows grouping lines of procedure in "chunks." A chunk is a group of lines in User Language that are compiled to produce

object code (quads) of a specific number of bytes. This number of bytes is known as the chunk size (*ch_size*).

Thus a REPORT STATE RUNG CHUNK 300 breaks up procedures into chunks that correspond to approximately 300 bytes of compiled code (quads). A user in a particular state while evaluating a procedure is always in exactly one chunk. A “chunk” report will present each chunk as the corresponding line numbers of the source procedure.

Chunks can never cross INCLUDE statement boundaries. In addition, chunks can be terminated by the end of a procedure. For this reason, some chunks in a CHUNK report might actually be smaller than the chunk size specified.

A chunk in a procedure will also generally be larger than the size specified on a RESOLUTION statement applicable to the procedure.

Since different User Language statements compile to different numbers of bytes, the actual size of the chunks will generally not be exactly the chunk size specified on the REPORT statement. For this reason, the actual size of each chunk is displayed on all CHUNK reports. Since the smallest possible User Language chunk is 4 bytes, a chunk size of 1 is equivalent to a chunk size of 4 and a chunk size of 30 is equivalent to a chunk size of 32.

IFJOB By originating IFAM2 jobname. This can be useful in determining which IFAM jobs are generating a lot of activity. This report is not likely to be of interest unless IFAM2 shows up as significant on a REPORT STATE ??? WHAT report.

IFCOMP By originating IFAM2 compilation name. Caution must be used in this report, because the same compilation name can be used by many different programs for many different purposes. This report is not likely to be of interest unless IFAM2 shows up as significant on a REPORT STATE ??? WHAT report.

IFJCOMP By originating IFAM2 compilation and jobname. This has an advantage over the IFCOMP report: it will not combine two identically named but different compilations for two different jobs. It has the disadvantage that if an identical compilation is used by two jobs, the totals for these compilations will not be combined.

This report is not likely to be of interest unless IFAM2 shows up as significant on a REPORT STATE ??? WHAT report.

IFFUNC By IFAM2 function name (IFFIND, IFPUT, etc.). This report is not likely to be of interest unless IFAM2 shows up as significant on a REPORT STATE ??? WHAT report.

IFCHUNK ch_size By pieces of IFAM2 load modules. Every IFAM2 load module can be broken up into an arbitrary number of pieces. *SirTune* allows grouping bytes of code in "chunks." A chunk is a range of offsets in the load module of a specific number of bytes. This number of bytes is known as the chunk size (*ch_size*). Thus a REPORT STATE RUNG IFCHUNK 256 breaks up load modules into chunks of 256 bytes.

This report lists usage by IFAM2 load module name and offset within the load module. For this report to be useful, the offsets withing the load modules must be converted to offsets within specific CSECTs. This can be done use a load map for the load module. Offsets within CSECTs must then be converted to source statement offsets, using compiler listings if a higher level language was used to generate the CSECTs.

This report is not likely to be of interest unless IFAM2 shows up as significant on a REPORT STATE ??? WHAT report.

This report will produce no useful data on CMS. Under MVS, the IFINTF object deck shipped in the *SirTune* object library must be linked with the IFAM2 load modules of interest for this report to produce useful information.

For example, to get a breakdown of running users by procedure being compiled, enter the following in SIRTUNEI or TUNERPTI:

```
REPORT STATE RUNG COMP
```

Or, using XML, specify this:

```
<report type="comp" state="rung">
```

Since a user is using CPU if and only if the user is running, that is, in state RUNG, this report gives an estimate of CPU usage by procedure being compiled.

To get a breakdown of swapping users by individual lines within a procedure, enter the following in SIRTUNEI or TUNERPTI:

```
REPORT STATE SWPG CHUNK 4
```

Since a chunk size of 4 is the smallest possible chunk size, this guarantees that no two lines of User Language will be grouped together in the report. Note that to produce this report, data must be collected for state SWPG by *SirTune*. This would require a

COLLECT statement ("COLLect state [extra_data]" on page 15) in SIRTUNEI for *SirTune*.

To get a breakdown of users waiting for journal I/O by evaluating procedure, enter the following in SIRTUNEI or TUNERPTI:

```
REPORT STATE WJRIO EVAL
```

To produce this report, data must be collected for state BLKIN by *SirTune*.

It is possible to specify several breakdowns on a single REPORT STATE statement. For example,

```
REPORT STATE WDISK WHAT EVAL CHUNK 1000
REPORT STATE WDISK EVAL CHUNK 100 CHUNK 10
```

requests breakdown of users waiting for disk I/O by general activity category (WHAT), procedure being evaluated, and chunks of 1000, 100, and 10 in the procedures being evaluated.

8.10 REPORT SUMMARY

The SUMMARY report provides a summary of the average number of users in each sample, broken down by user state and wait type.

If using XML to configure the report, the XML equivalent to a REPORT SUMMARY control-card statement is two elements:

```
<report type="state">
<report type="waittype">
```

This report is of particular interest in determining potential problem areas. For example, a large average for number of users in the WTSV (waiting for server) state might be indicative of a server shortage. And a large average for number of users waiting for checkpoint I/O might be indicative of problems with the checkpoint dataset.

Use caution when interpreting this report, especially when making negative conclusions (that is, deciding that something is not a problem area). For example, if WTSV does not show up on this report, it could mean one of two things:

- There are never any users waiting for server.
- No COLLECT statements were coded to tell *SirTune* to collect data for users in state WTSV.

8.11 REPORT SYSPARM

The SYSPARM report provides the values of all system and scheduler parameters in the Online associated with the sample dataset. This report provides identical output to a `VIEW SYSTEM CWAIT` command issued in the Online associated with the sample dataset.

This report is only available in the *SirTune* report writer versions 1.6 and later. In addition, the data presented by this report is only provided in sample datasets collected with the *SirTune* data collecter version 7.0 and later. For sample datasets created with earlier *SirTune* data collectors, the REPORT SYSPARM report will be empty.

While this is not one of the default reports produced by the report generator, it is recommended that this report always be included in report output, because it provides information about system settings that might prove useful in interpreting the other reports.

8.12 REPORT WHATC | WHATCM | WHATCS TOTAL | CHUNK ch_size

The WHATC/WHATCM/WHATCS reports break down *Model 204* CPU usage by *Model 204* "activity" and CSECT. The most common *Model 204* activities are evaluating and compiling. These reports are generally only of interest to *Model 204* internals experts, though they might be of interest in shops that have extensive in-house $function libraries.

The WHATC report breaks down all CPU usage by *Model 204* activity and CSECTs. Since the breakdown is a CSECT-level breakdown, the setting of the MPVIRT parameter ("MPVirt" on page 44) has no effect on the WHATCM and WHATCS reports — only the real task on which a request was running is used to distinguish CSECTS and CSECTM.

When running the MP/204 feature, the WHATCM report breaks down maintask CPU usage by activity and CSECT, and the WHATCS report breaks down subtask CPU usage by activity and CSECT. If not running the MP/204 feature or running the feature with 0 subtasks, WHATC is equivalent to WHATCM, and WHATCS is always 0.

Note: If using XML for the parameters ("Using XML input for report configuration" on page 53), a single convention can handle these different WHATC report variations: specify a "task=maintask" or "task=subtask" attribute for a "type=whatc" report to invoke, repectively, a WHATCM or a WHATCS report. For example, to get a WHATCS report:

```
<report type="whatc" task="subtask">
```

WHATC|M|S reports are further qualified by the TOTAL or CHUNK keyword:

- TOTAL requests a breakdown of CPU usage by *Model 204* activities and entire CSECT. For example, to produce a breakdown of CPU usage where each entry in the report is an entire CSECT/*Model 204* activity combination, you use:

  ```
  REPORT WHATC TOTAL
  ```

  Since TOTAL is the default, a `REPORT WHATC` statement is equivalent to `REPORT WHATC TOTAL`.

  If using XML in the report configuration input, `REPORT WHATC TOTAL` is equivalent to `<report type="whatc" chunk="total"/>`.

- CHUNK requests a breakdown of CPU usage by pieces of each CSECT for each *Model 204* activity. The size of each piece or chunk is *ch_size* bytes of object code. Thus to get a breakdown of CPU usage by 128-byte chunks of object code, code the following in SIRTUNEI or TUNERPTI:

  ```
  REPORT WHATC CHUNK 128
  ```

You can specify multiple breakdown types on a single REPORT WHATC statement. For example, this statement requests a breakdown of CPU usage by entire CSECT/*Model 204* activity combination, and by CSECT chunks of 512 bytes, and by chunks of 64 bytes:

```
REPORT WHATC TOTAL CHUNK 512 CHUNK 64
```

Note: The MAPCORE statement (“MAPcore” on page 43) in the report configuration input specifies that instructions outside of the loaded *Model 204* module should be treated as a “CSECT,” allowing analysis of CPU usage in the entire address space.

8.13 The TOP parameter

Many reports produce a listing of entities ranked in order of the number of samples for which data was found. The number of entities listed on these reports is either 50 or the value indicated on the last TOP statement before the REPORT statement. Thus, the following would list the top 50 evaluating procedures if no TOP statements appear in SIRTUNEI:

```
REPORT STATE RUNG EVAL
```

The first report below would list the top 10 evaluating procedures, while the second report would list the top 100:

```
TOP 1Ø
REPORT STATE RUNG EVAL
TOP 1ØØ
REPORT STATE RUNG EVAL
```

You can also explicitly specify the number of top entities to be listed on certain REPORT statements. The report statements on which this can be done are:

- REPORT CSECT | CSECTM | CSECTS
- REPORT DISKIO
- REPORT QUADC | QUADCM | QUADCS
- REPORT SERVIO
- REPORT STATE
- REPORT WHATC | WHATCM | WHATCS

For example, to list the top 100 evaluating procedures for state RUNG, simply code:

```
REPORT STATE RUNG EVAL TOP 1ØØ
```

To list the top 75 *Model 204* database file tables with waits on disk I/O, code:

```
REPORT DISKIO TABLE TOP 75
```

On REPORT statements with multiple breakdowns, the TOP statement must come after the breakdown description, and it applies only to the immediately preceding breakdown. Thus, the following is ***invalid***:

```
REPORT STATE RUNG TOP 1ØØ EVAL
```

The following would result in the top 60 chunks for the CHUNK 1000 breakdown, and the top 100 chunks for the CHUNK 100 breakdown:

```
TOP 6Ø
REPORT STATE RUNG CHUNK 1ØØØ CHUNK 1ØØ TOP 1ØØ
```

CHAPTER 9 Model 204 States

When the *SirTune* sampling program is collecting a sample it scans all logged on users. Each user is classified by its **state**. The user's state is a general indication of the type of activity occurring in a user thread. These states roughly correspond to the states reported by the *Model 204* performance monitor, though broken down to a finer level of detail.

The following primary states are distinguished by *SirTune*:

BLKIN This includes any user that is blocked, that is waiting for something, in a server and not waiting for user input. This is distinguished from BLKIU because waits for things other than user input are generally viewed as a performance problem while waits for user input are not.

BLKIU This includes any user that is blocked, that is waiting for something, in a server and waiting for user input. This is distinguished from BLKIN because waits for things other than user input are generally viewed as a performance problem while waits for user input are not.

BLKON This includes any user that is blocked, that is waiting for something, not in a server and not waiting for user input. This is distinguished from BLKOU because waits for things other than user input are generally viewed as a performance problem while waits for user input are not.

BLKOU This includes any user that is blocked, that is waiting for something, not in a server and waiting for user input. This is distinguished from BLKON because waits for things other than user input are generally viewed as a performance problem while waits for user input are not.

REDY This includes any user that is ready to run, that is, in a server and not waiting on anything but not actually being run. Generally a user is in state REDY because another user is currently running.

RUNG This includes any user that is running, that is, using CPU. Unless MP/204 is installed, there can never be more than one user in state RUNG per sample.

RUNGM If MP/204 is installed, this includes any user that is running, that is, using CPU, in maintask mode. There can never be more than one user in state RUNGM per sample. See “The RUNGM and RUNGS states” on page 79.

RUNGS If MP/204 is installed, this includes any user that is running, that is, using CPU, in subtask mode. See “The RUNGM and RUNGS states” on page 79.

SWPGI This includes any user that is in the process of being swapped into a server.

SWPGOBN This includes any user that is in the process of being swapped out of a server because it is waiting on something other than user input. If what the user was waiting on is still not completed at the point the user is swapped out, the user switches to state BLKON.

SWPGOBU This includes any user that is in the process of being swapped out of a server because it is waiting on user input. If what the user was waiting on is still not completed at the point the user is swapped out, the user switches to state BLKOU.

SWPGOW This includes any user that is in the process of being swapped out of a server because it is has been server sliced. If no servers of appropriate size are available at the point the user is swapped out, the user switches to state WTSV.

WPST This includes any PST that is not running.

WTSV This includes any user that is waiting for a server to become available so that the user could be run. The only reason a user would be in the WTSV state is that all servers of appropriate size are occupied by other users that cannot be swapped out of server.

In this list the term “waiting for user input” refers to a thread waiting for terminal or line input. In addition, a wait for a response to the console message issued by user 0 on a HALT command is also considered a user input wait. “Sleep” waits, that is, waits resulting from the *SLEEP command and the PAUSE statement, are not considered user input waits.

In addition to the above primary states, several composite states are provided for convenience and report generation. For example, composite state SWPG is made up of primary states SWPGI, SWPGOBN, SWPGOBU and SWPGOW. Thus any user in any of the indicated primary states is also considered to be in state SWPG. The following are the available composite states, their component primary states and an explanation that provides an intuitive feel for the meaning of the composite state.

ALL This is a composite state that includes all primary states. Any logged on user or PST is considered in state ALL.

ALLI This state is made up of RUNG, REDY, BLKIN and BLKIU. It includes any user currently in a server and not being swapped out. It does not include non-running PSTs.

ALLN This state is made up of RUNG, REDY, BLKIN, BLKON, WTSV, SWPGI, SWPGOBN and SWPGOW. It includes any user in not blocked for user input. It does not include non-running PSTs.

BLK This state is made up of BLKIN, BLKIU, BLKON, BLKOU, SWPGOBN and SWPGOBU. It includes any user that is blocked on anything.

BLKI This state is made up of BLKIN and BLKIU. It includes any user that is in a server and blocked on anything.

BLKN This state is made up of BLKIN, BLKON and SWPGOBN. It includes any user that is blocked for something other than user input.

BLKO This state is made up of BLKON and BLKOU. It includes any user that is not in a server but is blocked on something.

BLKU This state is made up of BLKIU, BLKOU and SWPGOBU. It includes any user that is waiting for user input.

OSERVN This state is made up of SWPGOBN and BLKON. It includes any user that is either not in a server or being swapped out of a server because it is blocked on something other than user input.

OSERVU This state is made up of SWPGOBU and BLKOU. It includes any user that is either not in a server or being swapped out of a server because it is blocked on user input.

OSERVW This state is made up of SWPGOW and WTSV. It includes any user that is either waiting for a server or being swapped out of a server so that it can wait for a server to free up. This latter case only happens when a user is server sliced.

REDYR This state is made up of RUNG and REDY. It includes any user that is not blocked on anything and is in a server. Users in state REDYR can either running or waiting for the *Model 204* scheduler to provide CPU to run.

RUNBL This state is made up of RUNG, REDY, WTSV and SWPGOW. It includes any user that is not blocked on anything, that is, is runnable. Users in state RUNBL can either running or waiting for the *Model 204* scheduler to provide the resources (CPU and/or server) to run.

SWPG This state is made up of SWPGI, SWPGOBN, SWPGOBU and SWPGOW. It includes any user that is being swapped into or out of a server.

SWPGO This state is made up of SWPGOBN, SWPGOBU and SWPGOW. It includes any user that is being swapped out of a server.

SWPGOB This state is made up of SWPGOBN and SWPGOBU. It includes any user that is being swapped out of a server because it is blocked on something.

Any of the above primary or composite states can be included on COLLECT statements for input to *SirTune* and REPORT STATE statements for input to SIRTUNER. Some valid COLLECT statements are:

```
COLLECT BLKN SWPG
COLLECT ALLN
COLLECT BLKIN BLKON SWPGOBN WTSV SWPGOW SWPGI
```

Some valid REPORT STATE statements are

```
REPORT STATE BLKIN EVAL
REPORT STATE SWPG CHUNK 100
REPORT STATE ALLN EVAL CHUNK 1000 CHUNK 4
```

In addition to user states, *SirTune*'s COLLECT statement allows you to request information about **DISKIO** and **CFR**. The following is a valid COLLECT statement

```
COLLECT DISKIO CFR
```

but there is no REPORT STATE statement which allows DISKIO nor CFR.

Any state requested in a REPORT STATE statement must have had the corresponding primary states explicitly or implicitly specified on COLLECT statements for *SirTune*. The simplest way to ensure this is by explicitly specifying any state to be used in a REPORT STATE statement on a COLLECT statement. For example, if one intends to produce the following reports with SIRTUNER:

```
REPORT STATE BLKN CHUNK 10
REPORT STATE SWPG CHUNK 10
```

one can code the following COLLECT statement for *SirTune*:

```
COLLECT BLKN SWPG
```

This statement is functionally equivalent to

```
COLLECT BLKIN BLKON SWPGOBN SWPGOBU SWPGOW SWPGI
```

In general, if running a relatively small ONLINE (an average of less than 20 logged on users)

```
COLLECT ALL
```

should not produce a prohibitively large amount of data and makes all reports possible. If running a midsize to large ONLINE (an average 20+ logged on users), the following

```
COLLECT ALLN BLKIU SWPGOBU
```

should collect a sufficient quantity of data to produce most interesting STATE reports without generating a prohibitively large sample dataset.

9.1 The RUNGM and RUNGS states

When running the MP/204 feature with *Model 204* a user that is in state RUNG can be further distinguished to be either running in maintask mode (RUNGM) or subtask mode (RUNGS) for the purposes of reporting. For example, the SIRTUNER statements

```
REPORT STATE RUNGM EVAL
REPORT STATE RUNGS EVAL
```

generate two reports. The first is a breakdown of users running in maintask mode by evaluating procedure and the second is a breakdown of users running in subtask mode by evaluating procedure. Maintask mode is often referred to as “serial” mode, and subtask mode is often referred to as “parallel” mode. The total observations for state RUNG in any sample is always equal to the total observations for state RUNGM plus the total observations for state RUNGS.

The distinction between maintask and subtask mode can be made either on the basis of the task on which a user is running (maintask or subtask), or on its virtual (or logical) MP mode (that is, whether it is capable of running in a subtask or not). The default distinction is made on the basis of the actual task on which a user is running. This can be changed with the SIRTUNER MPVIRT statement. This is generally the preferred setting when using the REPORT STATE RUNGM report to try to reduce the amount of maintask (serial) User Language code.

9.2 Wait types

Users in state BLK (blocked on anything), always have a wait type associated with them. These wait types are the same wait types that appear next to the users in a *Model 204* MONITOR command or in the SIRMON WAITTYP statistic. STATE reports can be requested by these wait types. To produce these STATE reports by wait type, COLLECT statements (collecting data for all states in which a wait type might occur) must be added to *SirTune*'s input stream (SIRTUNEI).

For example, disk I/O wait types are not swappable, so it is only necessary to collect state BLKIN to produce a REPORT STATE WDISK report. Since critical file resource waits are swappable, states BLKIN, BLKON, and SWPGOBN must all be collected to produce a REPORT STATE WCFREX report.

The available wait type reports along with the corresponding *Model 204* wait type number, a description of the wait type, and the required states to be collected are listed here:

WMISC 0 - Miscellaneous waits. Requires BLKN.

WDISK 1 - Wait for disk I/O. Requires BLKIN.

WUSERO 2 - Wait for user output. Requires BLKU.

WUSERI	3 - Wait for user input. Requires BLKU.
WOPERI	4 - Wait for operator input. Requires BLKU.
WDUMPO	5 - Wait for dump write. Requires BLKIN.
WDUMPI	6 - Wait for restore read. Requires BLKIN.
WENQUE	7 - Wait for miscellaneous enqueue. Requires BLKN.
WBUFF	8 - Wait for disk buffer. Requires BLKIN.
WPST	10 - Wait on PST. Requires BLKN.
WIFAM	11 - IFAM waits. Requires BLKN.
WSLEEP	12 - Waits for a time interval, including PAUSE and SLEEP statements. Requires BLKN.
WJRNLO	15 - Wait for journal output. Requires BLKIN.
WCHKPO	16 - Wait for checkpoint output. Requires BLKIN.
WWRITE	17 - Wait for a checkpoint DECB. Requires BLKIN.
WARBMO	18 - Waits for output arbitration. Requires BLKN.
WCHKPR	19 - Waits for a checkpoint request. Requires WPST.
WDISK	20 - Waits for checkpoint completion. Requires BLKIN.
WDEAD	21 - Wait forever (dead thread). Requires BLKU.
WVSAMI	22 - Wait for VSAM input. Requires BLKN.
WLOGIN	23 - Wait after login failure. Requires BLKN.
WCFREX	24 - Wait for critical file resource in exclusive mode. Requires BLKN.
WCFRSH	25 - Wait for critical file resource in share mode. Requires BLKN.
WVTBUF	26 - Wait for VTAM buffer. Requires BLKN.
WCONVI	27 - Wait for inter-process input. Requires BLKN.
WCONVO	28 - Wait for inter-process output. Requires BLKN.
WSCTYI	29 - Wait for security interface. Requires BLKN.

WS$WAI 30 - Swappable $WAIT call. Requires BLKN.

WN$WAI 31 - Non-swappable $WAIT call. Requires BLKIN.

WULDB2 32 - Wait for DB2 subtask. Requires BLKN.

Thus to produce a breakdown of disk I/O waits by evaluating procedure and by individual lines within the procedures, code the following in SIRTUNEI:

```
REPORT STATE WDISK EVAL CHUNK 4
```

To get a breakdown of waits for miscellaneous enqueues (including record locks) by evaluating procedure and by individual lines within the procedures, code the following in SIRTUNEI:

```
REPORT STATE WENQUE EVAL CHUNK 4
```

In addition to these primary wait types, there are a few composite wait types for which reports can be generated. These composite wait types, their component primary wait types, and a description of what the composite wait types measure are listed here:

WCFR This is made up of WCFREX and WCFRSH. It measures all waits on critical file resources whether for exclusive or share control.

WLOG This is made up of WJRNLO, WCHKPO, WWRITE, and WARBMO. It measures all waits on activities associated with logging for *Model 204* recovery, that is, all checkpoint and journal I/O related waits.

To get a breakdown of waits for critical file resources by evaluating procedure and by individual lines with the procedures, code the following in SIRTUNEI:

```
REPORT STATE WCFR EVAL CHUNK 4
```

9.3 Critical file resource states

Critical file resources are used by *Model 204* to provide multi-user concurrency control on a file level. This control mechanism will sometimes exacerbate some other performance bottleneck. A high value for number of users per sample with wait types CFREX and CFRSH in the SUMMARY report suggests that critical file resource enqueuing bears closer examination.

There are four different critical file resources:

DIRECT Protects table B updates and accesses.

INDEX Protects accesses and updates of table C and the ordered index.

EXISTS Protects accesses and updates of the existence bit map.

RECENQ Protects accesses and updates of the record enqueuing table. This is the only critical file resource that can be eliminated by the use of the FIND WITHOUT LOCKS User Language statement.

A first step to investigating a critical file resource enqueueing problem is to produce reports for STATE WCFR. This will help isolate the programs or lines of code that encounter frequent or long critical file resource waits. Probably the most useful report would be produced by this statement:

```
REPORT STATE WCFR CHUNK 4
```

This will break down critical file resource waits by individual lines of User Language code. Unfortunately, the problem with this type of analysis is that it focuses on the "victims" of critical file resource waits rather than the "culprits," the lines of code holding critical file resources causing other users to wait. While in some situations, the lines of code causing the critical file resource waits are the same lines that suffer from the waits, there is no way to be certain from the STATE WCFR report that this is indeed the case.

To determine the actual cause of critical file resource enqueuing, more data needs to be collected by the *SirTune* data collector. To have this additional data collected, simply specify the parameter CFR on a COLLECT statement for *SirTune*. This parameter can be specified alone or with other COLLECT parameters as in this statement:

```
COLLECT BLKN DISKIO CFR
```

After this additional CFR (Critical File Resource) data is collected, SIRTUNER is able to produce several additional reports to help isolate the cause of critical file resource enqueuing. The first report that might be useful is the CFRROOT report. This report indicates the base wait types that are behind critical file resource waits. The CFRROOT report does not provide information on which lines of code cause critical file resource waits, so it is not helpful for application tuning.

The CFRROOT report might indicate that application tuning (rather than system tuning) might be required to reduce critical file resource enqueuing. This would be indicated by a primary root cause of DISK (disk I/O waits) or maybe JRNLO (journal I/O waits).

You can attack a primary root cause for critical file resource waits either by trying to reduce overall disk I/O's or journal I/O's (with application tuning), or by specifically targeting those instructions that hold critical file resources.

To facilitate this latter option, several CFR states can be requested on SIRTUNER reports if CFR data had been collected by *SirTune*. These states are:

CFRHANY The state where a user holds any critical file resource.

CFRHDIR The state where a user holds the `DIRECT` critical file resource.

CFRHIND The state where a user holds the INDEX critical file resource.

CFRHEXS The state where a user holds the EXISTS critical file resource.

CFRHREC The state where a user holds the RECENQ critical file resource.

CFRBANY The state where a user holds any critical file resource and is preventing (blocking) another user from obtaining a critical file resource.

CFRBDIR The state where a user holds the DIRECT critical file resource and is preventing (blocking) another user from obtaining the DIRECT resource.

CFRBIND The state where a user holds the INDEX critical file resource and is preventing (blocking) another user from obtaining the INDEX resource.

CFRBEXS The state where a user holds the EXISTS critical file resource and is preventing (blocking) another user from obtaining the EXISTS resource.

CFRBREC The state where a user holds the RECENQ critical file resource and is preventing (blocking) an other user from obtaining the RECENQ resource.

It should be noted that the CFRB??? states are weighted based on the number of other users holding the resource and the number of users waiting for the resource. For example, if a user at a line of code holds the DIRECT resource and 3 other users are waiting for the resource, that line of code is considered to have 3 observations in the CFRBDIR state.

On the other hand, if a user at a line of code holds the DIRECT resource (in share mode) along with 4 other users, and a single user is waiting for the DIRECT resource, the line of code is considered to have 1/5th of an observation in the CFRBDIR state.

Generally, the most useful reports for reducing critical file resource waits are the CFRB reports. The statement

```
REPORT STATE CFRBANY CHUNK 4
```

will break down the state where a user is blocking another user from any critical file resource by lines of User Language code. This is probably the most useful of the STATE CFR???? reports. Once critical file resource blocking is isolated to specific User Language instructions, critical file resource enqueuing can be reduced by

- Reducing the number of times the offending instructions are executed.
- Reducing the amount of disk I/O performed by the offending instructions.
- Reducing the amount of CPU used by the offending instructions.

It might be tempting to use the FIND WITHOUT LOCKS User Language statement to reduce the critical file resource enqueuing associated with a statement. This will only work if the resource causing conflicts is the RECENQ resource. All other critical file resources are processed exactly the same way, whether or not a locked record set is being used.

However, if the resource causing the conflict is indeed the RECENQ resource, it is still *not* recommended that the solution be FIND WITHOUT LOCKS. A high conflict rate on the RECENQ resource indicates that the environment has a high update activity level, which means that operating on unenqueued found sets is a questionable tactic at best. A high conflict rate on the RECENQ resource might suggest examination of strategies for releasing found sets before any terminal I/O occurs.

The CFRH??? reports can be useful for tracking potential critical file resource enqueuing problems (perhaps in a test environment) before they actually happen. These states include any user that holds a critical file resource, whether or not it is blocking anyone. These reports are difficult to interpret, however, since they require a fairly good estimate of expected future usage patterns to have any predictive value.

CHAPTER 10 Model 204 Quad Types

The basic unit of work in compiled User Language is the “quad.” Every User Language statement generates one or more quads. By breaking down *Model 204* processing by quad types (using the QUAD parameter on STATE reports), one can get an idea of the general type of work occurring in an Online. To interpret a QUAD report, it is important to know what the individual quads do. $Function quads are easy to interpret since they are listed by function name. Other quads are listed here with a brief description of their function.

ADDAV Add a fieldname/value pair to a record. Can be an ADD statement, part of a CHANGE statement or one line in a STORE RECORD statement.

INSERT Add a fieldname/value pair to a record for the INSERT statement.

ANDQ Perform a logical AND in a 204 expression (possibly in an IF statement).

ARADD Perform an arithmetic addition in an expression.

ARDIV Perform an arithmetic division in an expression.

ARMUL Perform an arithmetic multiplication in an expression.

ARSUB Perform an arithmetic subtraction in an expression.

ASSQ Assigns result (possibly intermediate) of an expression to a result variable.

ASSQCN Assigns integer constant to a (possibly intermediate) fixed result variable.

ASSQEE Assigns result (possibly intermediate) of a string expression to a string result variable.

ASSQNN Assigns result (possibly intermediate) of a numeric expression to a numeric result variable.

ASSQO Assigns Blob field value to the user buffer.

AUDITUS The part of the AUDIT statement that actually moves data into the journal.

BACKOUT The BACKOUT statement.

BAND	Boolean AND. Combine two bit maps as part of FIND statement or in list manipulation.
BASE	Handles the SFGE$ and SFL$ conditions in a FIND statement.
BEQVAL	Handles the *field* `= VALUE IN` *label* condition for an ordered character field, where *label* refers to an enclosing FOR EACH VALUE loop or to a NOTE statement.
BLIKE	Handles the IS LIKE condition on a FIND statement for an ordered character field.
BNOT	Boolean NOT. Invert a bit map as part of FIND statement or in list manipulation.
BOPER	The part of a FIND statement associated with looking up a a fieldname/value pair in the ordered index. Never occurs in a group context FIND.
BOR	Boolean OR. Combine two bit maps as part of FIND statement or in list manipulation.
BRANGE	Handles a range find condition for an ordered index field on a FIND statement.
BRNCH	Branch used in computed JUMP statement or to branch around subroutine.
BVAL	The FIND ALL VALUES statement.
CHGAV	Delete a fieldname/value pair from a record for the CHANGE statement.
CLGLOB	The CLEAR GLOBALS statement.
COMMIT	The COMMIT statement.
COUNT	Count number of entries in found set. Used in COUNT and FIND AND PRINT COUNT statements. Also used in FIND statement to determine if a DYRWT message should be issued for a table B (direct) search.
CSORT	Copy data to CCATEMP for SORT or FOR EACH VALUE IN ORDER statement.
CTO	The COUNT OCCURRENCES statement.
DEDB2	The EXEC DB2 ... END EXEC statement.

DEQSET	Unlock a record set after deleting them in a DELETE RECORDS IN/ON statement.
DIS	Handles IS =, IS ALPHA =, and IS NUM = conditions on a FIND statement.
DLIKE	Handles a IS LIKE condition on a FIND statement for a non-ordered character field.
DOPER	Part of a Find that looks up a fieldname/value pair in table C, the ordered index, or table B (direct search or hash key). The table C, ordered index, and hash key lookups show up as DOPER instead of POPER/NOPER, BOPER/NOPER, and HOPER/NOPER pairs in a group context find. The only time DOPER shows up in a non-group context is when a table B (direct) search is performed.
DOTEST	Increments and tests index in a FOR index loop (FOR I FROM ...)
DPRES	Handles an IS PRESENT condition on a FIND statement.
DRANGE	Handles a range find condition for an non-ordered field on a FIND statement.
DROPA	Deletes each occurrence of a field in a record (DELETE EACH statement).
DROPAV	Delete a fieldname/value pair from a record. Used for the DELETE statement.
D2DELRS	A DELETE RECORDS statement on a remote file.
D2EFIND	A FIND statement on a remote file.
D2ESORT	A SORT statement on a remote file.
D2FILR	A FILE RECORDS statement on a remote file.
D2LIST	A PLACE RECORDS or REMOVE RECORDS statement on a remote file.
D2OPCUR	“Open cursor” on a remote file, to fetch sorted records or values to handle FOR EACH RECORD/VALUE in remote group context.
ECALL	The CALL statement.
EICLOSE	The CLOSE statement.
EICLRGO	Handles CLEAR GLOBAL statement for global images.

EICLRT	The CLEAR TAG statement.
EIMOD	The MODIFY statement.
EIOPEN	The OPEN statement.
EIPOS	The POSITION statement.
EIPREP	The PREPARE SCREEN statement. This includes PREPARE statements where the word `SCREEN` is implied.
EIPREPI	The PREPARE IMAGE statement. This includes PREPARE statements where the word `IMAGE` is implied.
EIREAD	The READ SCREEN statement. This includes READ statements where the word `SCREEN` is implied.
EIREADI	The READ IMAGE statement. This includes READ statements where the word `IMAGE` is implied.
EIRPOS	The RELEASE POSITION statement.
EITAG	The TAG statement.
EIWRITE	The WRITE IMAGE statement. This includes WRITE statements where the word `IMAGE` is implied.
ELCLOSE	The CLOSE PROCESS statement.
ELCNFRM	The CONFIRM statement.
ELCNFRMD	The CONFIRMED statement.
ELFLUSH	The FLUSH statement.
ELINVITE	The INVITE statement.
ELOPEN	The OPEN PROCESS statement.
ELQUERY	The QUERY PROCESS statement.
ELRCV	The RECEIVE statement.
ELSEND	The SEND statement.
ELSIGNL	The SIGNAL statement.
ELSNDERR	The SEND ERROR statement.

ELTEST	The TEST statement.
ELTRNSFR	The TRANSFER statement.
ELWAIT	The WAIT statement.
EMID	The IDENTIFY statement.
ENQSET	Lock a record set about to be deleted in a DELETE RECORDS IN/ON statement.
ENRA	Extract the value of a field. Mostly used for IS LIKE and IS NOT LIKE clauses.
ENRACO	Copies a value in a field used in an expression into STBL.
ENRAPR	Print a field value for a PRINT statement.
ENRASSF	Assign the value of a field to a percent variable.
EQQ	Perform a numeric or string equality test to produce a logical true or false (0 or 1) result (possibly in an IF statement).
ERETRN	The RETURN, BYPASS PENDING, or RETRY statements and implied RETURNs at the end of a subroutine.
EVCRLOC	IFAM only.
EVIFONT	Font switch produced by `*FONT` keyword on a PRINT statement. Only used in DBCS environment.
EVOPCUR	IFAM only.
FEO	Finds next occurrence of a field for FOR EACH OCCURRENCE statement.
FILEDOL	Handles the FILE$ in a group context FIND statement.
FMCMD	File maintenance commands (DEFINE FIELD, REDEFINE FIELD, DELETE FIELD, RENAME FIELD, SECURE, DESECURE, ALLOCATE, DELETE GROUP). Only allowed to be used by CCA products.
FOR	FOR EACH RECORD, FOR n RECORDS, and FOR EACH VALUE loops. Finds the next record number or value to process in loop.
FOREMOTE	Get next record in a FOR EACH RECORD or FOR EACH VALUE loop against a remote file.
FOREV	The start of a REPEAT FOREVER or REPEAT WHILE loop.

FORNO	The FOR RECORD NUMBER statement.
FORNOREM	The FOR RECORD NUMBER statement against a remote file.
F2LKI	FLOD only.
GETMORE	IFAM only.
GETNN	IFAM only.
GLOOP	Group LOOP. Switch to next file in group in any group related operation including FIND statements.
HOPER	Hash OPERation. Looks up records in table B in a hash key file when the hash key appears in a FIND statement.
INTQ	Converts a string or floating point value in an expression to a fixed-point value.
ISNTPRES	Check for absence of a field in a record for an IS NOT PRESENT test in an expression to produce a logical true or false (0 or 1) result (possibly in an IF statement).
ISORT	Initialization for SORT or FOR EACH VALUE IN ORDER statement.
ISPRES	Check for presence of a field in a record for an IS PRESENT test in an expression to produce a logical true or false (0 or 1) result (possibly in an IF statement).
ITSQ	Used for SoftSpy.
JUMP	Branch associated with JUMP TO or IF/THEN/ELSE statement.
JUMPTEST	Branches to correct computed jump label in computed JUMP statement.
LIKE	Perform a wildcard character test for an IS LIKE clause in an expression to produce a logical true or false (0 or 1) result (possibly in an IF statement).
LOCDOL	Evaluate LOCATION$ FIND condition for Parallel Query Option/204.
LOOP	Get the next file segment in a multi-segment statement. Used in FIND, FILE RECORDS, PLACE RECORDS, REMOVE RECORDS, and DELETE RECORDS statements.
LOOPEND	Branch associated with LOOP END statement.
LOOPENDC	Conditionally exit a loop for a REPEAT WHILE statement.

MSORT	Sort data in CCATEMP for SORT or FOR EACH VALUE IN ORDER statement.
NEGQ	Unary minus sign, for example, `%I=-%I`.
NEWPAGE	The NEWPAGE statement.
NMEVEQ	Evaluates equality condition for numeric range field on a FIND statement.
NMEVGE	Evaluates greater than or equal condition for numeric range field on a FIND statement.
NMEVGT	Evaluates greater than condition for numeric range field on a FIND statement.
NMEVLE	Evaluates less than or equal condition for numeric range field on a FIND statement.
NMEVLT	Evaluates less than condition for numeric range field on a FIND statement.
NOPER	Normalize OPERation. Converts a single record entry, record list (in table D) or a bit map (in table D) into a bit map in CCATEMP. Part of a FIND statement.
NOTDSET	Inverts a found set or list for the DELETE RECORDS statement.
NOTE	Note a value for the NOTE statement.
NOTQ	Perform a logical NOT in a *Model 204* expression (possibly in an IF statement).
NTIMES	The start of a `REPEAT n TIMES` loop.
NUMQ	Converts a string value in an expression to a floating point value.
ONQ	Sets ON unit address and branches around ON unit for the ON statement.
ORQ	Perform a logical OR in a *Model 204* expression (possibly in an IF statement).
PAFN	Prints all field names for the PRINT ALL FIELD NAMES (PAFN) statement.
PAUSE	The PAUSE statement.
PID	Prints the record ID for the `*ID` clause of a PRINT or AUDIT statement.

PLAREM	PLACE/REMOVE RECORDS ON/FROM LIST statement. Turn bits on/off in list bitmaps.
PNUM	Print a number; used in PRINT and FIND AND PRINT COUNT statements.
POINT	Evaluates POINT$ condition in a FIND statement.
POPER	The part of a FIND statement associated with looking up a a fieldname/value pair in table C. Never occurs in a group context FIND.
POSNEV	The POSITION statement.
POSQ	Unary plus sign, for example, `%S=+%S`.
PRINTALL	The PRINT ALL INFORMATION (PAI) statement.
PRTNCOL	Prints a field value in the PRINT statement if the fieldname was immediately preceded by a number indicating a maximum number of lines.
PSNOP	Copying a bit map page. Use in PLACE RECORDS and REMOVE RECORDS statements and in FIND statements on previously created found sets or lists.
PSNOPEBM	Part of FIND statement; copies (sometimes) existence bit map to CCATEMP.
PSTRG	Print a string; used in PRINT statements.
PTALL	EACH fieldname parameter in a PRINT statement.
QEND	End of most multi-part or multiple file segment statements. Occurs in FIND, DELETE RECORDS IN, SORT, FILE RECORDS, PLACE, REMOVE, and many other statements.
QENDF	Ends a FOR loop.
QENDG	End of group loop. Occurs in FIND and other statements in group context. Associated with a GLOOP quad.
RELALL	The RELEASE ALL RECORDS statement.
RELSERQ	The RELEASE RECORDS IN statement.
REMBREV	The REMEMBER statement.
RESHTR	The RESET HEADER/TRAILER statement.

RRECD	Remove a record from table B for a DELETE RECORD statement.
RSPOPER	The part of a FIND statement associated with looking up a fieldname/value pair in table C. Never occurs in a group context FIND. RSPOPER is used instead of POPER when using record security in an APSY pre-compiled proc.
SAVE	Save invisible key data in table C or in the ordered index for a FILE RECORDS statement.
SETHIO	Start of SET HEADER/TRAILER statement.
SITEM	Start of STORE RECORD statement. Creates empty record in table B and adds hash or sort key in hash or sort key file.
SITEMREM	Start of STORE RECORD statement for remote file.
SOLOOP	Not used.
SOPER	Set OPERation. Sets a segments bit map in a found set for a FIND statement (or a PLACE/REMOVE statement). Also responsible for enqueuing on the found set unless doing FIND WITHOUT LOCKS.
SOPEREBM	Part of DELETE RECORDS statement; copies existence bit map to CCATEMP.
SOR	Find next record in a SORT KEY file for a FOR EACH RECORD IN ORDER BY sort key statement.
STOREND	Not used.
STPIF	The STOP IF COUNT statement.
STRGQ	Converts a floating point value in an expression to a string value.
SUBS	Retrieves a value in an array based on a subscript.
TAB	TAB parameter in a PRINT, SET HEADER, or SET TRAILER statement.
UITEM	Start of UPDATE RECORD statement.
UITEMREM	Start of UPDATE RECORD statement for remote file.
WCARDX	Output a PRINT or AUDIT line.
WCARDW	SKIP n LINES, where n > 1.

WITHQ	WITH parameter in a PRINT, AUDIT, SET HEADER, or SET TRAILER statement.

CHAPTER 11 *Wildcard Strings in SirTune and SIRTUNER Statements*

There are several *SirTune* statements that allow the use of wildcard strings to specify a set of matching strings with a single string. These statements and the associated component of *SirTune* are:

AUTHORIZE	SIRTUNE
RESOLUTION	SIRTUNER or SIRTUNEREPORT
SIRTUNED	A CMS command

The special characters and their meanings in wildcard strings are:

***** Matches any group of characters including a null string.

? Matches any single character.

" Indicates that the next character in the wildcard string is to be treated literally, even if it is a double-quotation mark ("), asterisk (*), or question mark (?).

For example, to match any string, use:

```
*
```

To match ABCDEFG and no other strings, use:

```
ABCDEFG
```

To match ABC, ABCXXX, ABC22, ABCDEFG, etc. and not match ABXCD, CAB, etc., use:

```
ABC*
```

To match XYZ, AXYZ, XYZA, AXYZB, etc. and not match XAYBZ, XYAZ, etc., use:

```
*XYZ*
```

To match ABC1, ABCD, ABCZ etc. and not match ABC, ABCDE, ABXC, etc., use:

```
ABC?
```

To match any string with exactly four characters and not match anything else, use:

```
????
```

To match the string ABC* and nothing else, use:

```
ABC"*
```

To match the string ABC? and nothing else, use:

```
ABC"?
```

To matches the string ABC" and nothing else, use:

```
ABC""
```

To match ABC, ABD, A1BC, A123B?, etc. and not match ABCD, XABC, AXXXXX, etc., use:

```
A*B?
```

CHAPTER 12 Estimating SIRTUNED Size Requirements

It is recommended that you do not spend a lot of time trying to size SIRTUNED, because the consequences of under- or over-estimating the SIRTUNED space requirements are relatively benign. However, if you want to size SIRTUNED, this chapter provides some basic rules of thumb for estimating the correct size.

12.1 A formula for the estimate

The size of SIRTUNED is mainly determined by these factors:

- The number of lines of compilations saved. This is the number of lines in pre-compiled APSY procedures compiled in the run, or if the ALLCOMP statement is specified for *SirTune*, the number of lines of all compiled procedures. In any case, a line counts as 1 each time it is compiled.
- The number of samples collected.
- The average number of users for which data is collected per sample.

To estimate the number of lines of compilations saved when the ALLCOMP statement has not been specified, and when subsystems are not START'ed and STOP'ed multiple times in a run, simply total the number of lines in all pre-compiled procedures in START'ed subsystems. To get an estimate of this:

1. Count the total number of pre-compiled procedures.
2. Estimate the average number of lines per pre-compiled procedure by entering the editor for a representative sample of them.
3. Multiply these two values.

Call the number of compiled lines `COMP_LINES`.

To estimate the number of samples, divide the number of seconds over which data is to be collected, by the sample interval length (1 or whatever was specified on the INTERVAL statement). Call this value `NUM_SAMP`.

The best guide to estimating the number of users for which data is collected per sample is the *Model 204* performance monitor. There are certain worst case values that one can assume, however:

- If only state RUNG is being collected, at most 1 user will have data collected per sample, unless MP/204 is installed (in which case, the upper limit is 1 plus the number of subtasks).

- If only REDY, RUNG, BLKIN, BLKIU, SWPGI, SWPGOW, SWPGOBN, and/or SWPGOBU states are being collected, the upper limit is the number of servers.
- Otherwise, a crude upper limit is the number of users.

Call whatever value one comes up with AVG_USERS.

The total number of bytes required for SIRTUNED can be estimated by this:

```
100,000 + ( 12 * COMP_LINES ) +
( NUM_SAMP * 64 * ( 1  + AVG_USERS ) )
```

This estimator provides a fairly generous estimate without being excessive. To determine the number of disk tracks required, divide the number of bytes produced by this estimator by the number of bytes per track at the block size for SIRTUNED (46,952 is the default bytes per track on a 3380, and 55,996 is the default on a 3390).

12.2 An example estimate

Suppose a shop has 100,000 lines of pre-compiled User Language code, expects to collect samples over 8 hours at 1 per second, is collecting data for states RUNG, REDY, and BLKIN, and has 30 servers defined in the ONLINE. These factors are clear:

```
COMP_LINES = 100,000

NUM_SAMP = 8 * 60 * 60 / 1 = 28,800
```

Since all the users in a collected state must be in a server, use the number of servers as a gross estimator for AVG_USERS, that is:

```
AVG_USERS = 30
```

This means that the estimated space requirements in this case is:

```
100,000 + ( 12 * 100,000 ) +
28,800 * 64 * ( 1  + 30 ) = 58,439,200
```

If the sample data was going to a 3380 with the *SirTune* default block size, it would require 1245 tracks or 83 cylinders.

The number of samples collected in this example is fairly extreme: generally there is not much benefit to collecting more than 10,000 samples. If sampling is limited to some key hours to restrict the number of samples collected to 10,000, the estimator becomes:

```
100,000 + ( 12 * 100,000 ) +
10,000 * 64 * ( 1  + 30 ) = 21,140,000
```

If the sample data in this case was going to a 3380 with the *SirTune* default block size, it would require 451 tracks or 31 cylinders.

APPENDIX A SirTune Data Collector Messages

The following messages are issued by the data collection part of *SirTune* (in version 1.5 or earlier, the SIRTUNE load module).

TUNE0001 SIRTUNE version *version* started *date time* on CPU *cpu* at site *site*.

This indicates the version of *SirTune* that is running (*version*), the date and time at which it started (*date time*), the CPU ID that is running (*cpu*) and the site ID of the customer (*site*).

TUNE0002 Invalid command *command*.

This indicates that an invalid statement was found in SIRTUNEI, or an invalid command was issued via SMSG or MODIFY. If the error was in SIRTUNEI, the line with the invalid statement is echoed before this message, and *SirTune* will not come up. In this case SIRTUNEI must be corrected before the **ONLINE** or **BATCH204** job will come up.

TUNE0003 Unable to open SIRTUNED.

This indicates that *SirTune* was unable to open the file SIRTUNED. This probably means that no DD statement was coded for SIRTUNED.

TUNE0004 Invalid format for SIRTUNED.

This indicates that the SIRTUNED has an invalid format. This could mean that SIRTUNED does not have record format VB or that the LRECL is less than 512. Correct the DD card to specify a valid format.

TUNE0005 Parameter missing for *cmd* command.

This indicates that a parameter was missing for a statement in SIRTUNEI, or a parameter was missing for a command issued via SMSG or MODIFY. If the error was in SIRTUNEI, the line with the invalid statement is echoed before this message, and *SirTune* will not come up. In this case, SIRTUNEI must be corrected before the **ONLINE** or **BATCH204** job will come up. In any case, the correct format of the statement or command can be found in “Configuration Statements for the Data Collector” on page 13 or “MODIFY and SMSG commands” on page 27.

TUNE0006 **Invalid *cmd* parameter - *parm*.**

This indicates that an invalid parameter was found for a statement in SIRTUNEI or for a command issued via SMSG or MODIFY. If the error was in SIRTUNEI, the line with the invalid statement is echoed before this message, and *SirTune* will not come up. In this case, SIRTUNEI must be corrected before the **ONLINE**, or **BATCH204** job will come up. The correct format of the statement or command can be found in "Configuration Statements for the Data Collector" on page 13 or "MODIFY and SMSG commands" on page 27.

TUNE0007 **Invalid *inc_exc* range *r_val*.**

This indicates that an invalid range was specified on an INCLUDE or EXCLUDE statement in SIRTUNEI. The line with the invalid statement is echoed before this message, and *SirTune* will not come up. SIRTUNEI must be corrected before the **ONLINE** or **BATCH204** job will come up. In any case, the correct format of the statement or command can be found in "Configuration Statements for the Data Collector" on page 13.

TUNE0008 **Getmain request failed, RC=*ret_code*. SIRTUNE terminated.**

This indicates that there was insufficient storage to initialize *SirTune*. Increase the amount of storage allocated to the region, using the REGION parameter on the EXEC card under MVS or using the DEFINE STORAGE command under CP.

TUNE0009 **Unable to load *loadmod*.**

This indicates that *SirTune* was unable to load the *Model 204* load module called ***loadmod***.

- Under MVS, this means the indicated load module is not in any of the STEPLIBs for the *SirTune* step. This could indicate that the *Model 204* load module is not called **ONLINE**, and a PGM statement is required, the load module name was misspelled on the PGM statement, or the STEPLIB statement was coded incorrectly.

- Under CMS this means that the indicated load module is not on any accessed disk. This could indicate that the *Model 204* load module is not called `M2Ø4ONLN`, and a PGM statement is required, the load module name was misspelled on the PGM statement, or the disk with the *Model 204* load module was never accessed or has been released.

TUNE0010 ***loadmod* did not complete initialization with SIRTUNE.**

This indicates that the *Model 204* load module did not make it through initialization. This means that no *SirTune* samples are collected. Check the output from the *Model 204* run to determine the problem(s), and correct them.

TUNE0011 Unable to determine release of *loadmod* (reason *hex_info*).

This indicates that *SirTune* could not determine the release of the *Model 204* load module. This could be because the load module ***loadmod*** is not a *Model 204* load module, or because it is an unsupported release of *Model 204*. If it is the latter case and it is impossible to switch to another release of *Model 204*, contact Sirius Software product support.

TUNE0012 *vm_id* is not logged on.

This message, which only occurs under CMS, means that the SIRTUNED service machine is not logged on. Either log the SIRTUNED service machine on, or correct the CMSOUT statement in SIRTUNEI if ***vm_id*** is not the name of the SIRTUNED service machine.

TUNE0013 *vm_id* is not ready for IUCV.

This message, which only occurs under CMS, means that the SIRTUNED service machine is not initialized. Either intialize the SIRTUNED service machine, or correct the CMSOUT statement in SIRTUNEI if ***vm_id*** is not the name of the SIRTUNED service machine.

TUNE0014 Maximum IUCV connections exceeded.

This message, which only occurs under CMS, means that the *Model 204* service machine with *SirTune* has exceeded its maximum allowed IUCV connections. Increase the value for OPTION MAXCONN in the user directory entry for the *Model 204* service machine.

TUNE0015 Maximum IUCV connections exceeded for *vm_id*.

This message, which only occurs under CMS, means that the SIRTUNED service machine has exceeded its maximum allowed IUCV connections. Increase the value for OPTION MAXCONN in the user directory entry for the SIRTUNED service machine.

TUNE0016 Not authorized to connect to *vm_id*.

This message, which only occurs under CMS, means that the *Model 204* service machine is not authorized to connect to the SIRTUNED service machine via IUCV. Add an appropriate IUCV entry to the user directory entry for either the *Model 204* service machine or the SIRTUNED service machine to allow IUCV communications between the two.

TUNE0017 Sampling set to *setting*.

This is an informational response to the SAMPLE command issued via MODIFY or SMSG that indicates the new sampling setting.

TUNE0018 **Sampling terminating.**

This is an informational response to the STOP command issued via MODIFY or SMSG that indicates that sampling is terminating.

TUNE0019 **Sample data set closed.**

This is an informational response to the CLOSE command issued via MODIFY or SMSG that indicates that the sample dataset has been closed and can be processed with SIRTUNER. Data continues to be collected to the sample dataset.

TUNE0020 **Sampling is *mode* - *num* samples collected.**

This is an informational response to the STATUS command issued via MODIFY or SMSG that indicates the current state of sampling and the number of samples that have been collected.

TUNE0021 **Sampling is already terminated.**

This is a response to the STOP command issued via MODIFY or SMSG that indicates that sampling has already been terminated. Sampling could have been terminated either because of a previous STOP command or because of a severe error in *SirTune*.

TUNE0022 **Not authorized.**

This is a response to any command issued via MODIFY or SMSG that indicates that the issuing user is not allowed to issue such commands. Under MVS, this means no users are authorized to issue MODIFY commands. For a user to get privilege to issue SMSG or MODIFY commands, an appropriate AUTHORIZE statement must be placed in SIRTUNEI.

TUNE0023 **SirTune Expired... *date*.**

This message appears when trying to bring up the *SirTune* data collection module after a trial period has expired. The message shows the date of expiration of the trial. Call Sirius Software support for a new expiration ZAP if the trial has been extended or *SirTune* has been purchased. Otherwise, deinstall *SirTune*.

TUNE0024 **SIRTUNE not authorized for CPU *cpu_id*.**

This message appears when trying to bring up the *SirTune* data collection module on a CPU for which it is not authorized. Call Sirius Software support for a new CPU ZAP if the new CPU is a replacement for an old one. Note that processor upgrade fees might be required for a new CPU. The ***cpu_id*** indicated in this message is the processor ID that should be given to Sirius Software for a new CPU ZAP.

TUNE0025 **Task *task_num* - *status* at *address*.**

This message is received in response to the MONITOR command for each task in the **ONLINE**. ***task_num*** is 0 for the *Model 204* maintask and the subtask number if it is an offload subtask for the MP/204 feature. ***status*** will be either `WAITING` or `RUNNING`, and ***address*** will be either an absolute address in the *Model 204* address space or a CSECT name and offset. If ***status*** is `RUNNING`, this message will be followed by message TUNE0026.

TUNE0026 **User *user_num* (*userid*) What = *what* Last = *last* Proc = *proc*.**

This message is received in response to the MONITOR command for each task in the **ONLINE**. It is always preceded by message TUNE0025. ***user_num*** and ***userid*** identify the user running in the **ONLINE**. ***what*** and ***last*** identify the current user activity, and ***proc*** identifies the procedure currently being run.

TUNE0027 **Invalid user number - *user_num*.**

This message is received in response to a RESTART or BUMP command. It indicates that the user number specified on one of these commands was either not a non-negative integer or it was greater than NUSERS for the **ONLINE**.

TUNE0028 **User *user_num* not logged on.**

This message is received in response to a RESTART or BUMP command. It indicates that the user number specified on one of these commands was inactive. That is, no user was logged on to the indicated user number.

TUNE0029 **User *user_num* not running.**

This message is received in response to the RESTART command. It indicates that the user number specified on one of these commands was not running.

TUNE0030 **Invalid abend code *code*.**

This message is received in response to the RESTART command. It indicates that a non-hexadecimal abend code was specified, or the abend code was greater than hexadecimal FFF.

TUNE0031 **0C1-0CF are the only valid abend codes under CMS.**

This message is received in response to the RESTART command. It indicates that an abend code less than 0C1 or greater than 0CF was specified to a *SirTune* running under CMS. Only abend codes between 0C1 and 0CF are valid under CMS.

TUNE0032 **User *user_num* bumped.**

This message is received in response to the BUMP command. It indicates that *SirTune* has indicated to *Model 204* that user number ***user_num*** should be “bumped.”

TUNE0033 **User *user_num* restarted.**

This message is received in response to the RESTART command. It indicates that *SirTune* has simulated an abend for the user number ***user_num***.

TUNE0034 **Invalid task number - *task_num*.**

This message is received in response to a RESTART command. It indicates that the task number specified on one of these commands was not an integer, was negative, or was greater than NMPSUBS for the **ONLINE**. If the MP/204 feature is not being used, the task number must always be 0 (the maintask number).

TUNE0035 **Task *task_num* restarted.**

This message is received in response to the RESTART command. It indicates that *SirTune* has simulated an abend for the indicated task number.

TUNE0036 **Not APF authorized, can't do restart.**

This message is received in response to the RESTART command. It indicates that the *SirTune* load module is not APF authorized, hence it cannot issue simulated abends under MVS. See "MVS installation" on page 116 for more information.

TUNE0037 **Not running in Model 204, can't restart.**

This message is received in response to the RESTART command, and it only occurs under CMS. It indicates that the address space was not running in the *Model 204* load module at the time the RESTART command was issued. This means that the virtual machine was probably running CMS system code, hence it is extremely unlikely to successfully intercept the RESTART command. Issue the RESTART command several times (until this message is not received) to try to catch the virtual machine in the *Model 204* load module. If this proves impossible, more drastic measures must be taken. These measures might include logging on to the virtual machine, issuing a CP VMDUMP command, and logging off.

TUNE0038 **Call Sirius Software for a new authorization.**

This message indicates that the authorization zap applied to *SirTune* is inconsistent. Probably, it was not applied correctly, or you have attempted to apply an authorization zap for an older version of *SirTune*. If this is preceded by message TUNE0040, follow the steps outlined for that message. Otherwise, please call Sirius Software.

TUNE0039 **Authorization zap produced *date time*.**

This informational message indicates the date and time that the *SirTune* authorization zap was created at Sirius Software. It is shown as *date time*, and it may help determine if the proper zap was applied.

TUNE0040 **Invalid checksum in authorization zap.**

This message indicates that the *SirTune* authorization zap was incorrectly entered. If the zap was manually entered, double check the contents from the original copy. If the zap was received electronically from Sirius Software, either it was modified during receipt or it was incorrectly transmitted. You should double check your steps for receiving it, or obtain a new zap from Sirius Software.

TUNE0041 ***PTCH usage*.**

This informational message indicates the offsets in the PTCH CSECT which have been changed from the initial value of zero, which approximates the maintenance level of the executing load module. *PTCH usage* can have one of the following forms:

- **No PTCH used** , which indicates that the entire patch space is zero.
- **PTCH usage:** *beg-end ...* , which indicates the ranges (in hexadecimal) in the patch area that have been changed from zero.

TUNE0042 **SIRTUNE *Expiration status*.**

This informational message indicates when, if ever, *SirTune* will expire on the current CPU. ***Expiration status*** will be one of the following:

Permanently authorized	This indicates that *SirTune* has been purchased, and it will never expire on the current CPU.
Expires mm/dd/yy	This indicates that *SirTune* is under a trial or rental agreement for the current CPU, and it will expire at the date indicated.

APPENDIX B SirTune Report Writer Messages

The following messages are issued by the report generation part of *SirTune* (the SIRTUNER load module). After any of these errors, SIRTUNER processing is terminated and no reports are produced.

TUNR0002 Invalid command *cmd*.

This indicates that an invalid statement was found in SIRTUNEI. The line with the invalid statement is echoed before this message. Valid SIRTUNER statements are listed in "Configuring the Report Generator" on page 37.

TUNR0003 Unable to open *ddname*.

This indicates that *SirTune* was unable to open the indicated ***ddname***. This error can occur for ddnames SIRTUNED or SIRTUNEO. This probably means that no DD statement was coded for SIRTUNED or SIRTUNEO.

TUNR0004 Invalid format for SIRTUNED.

This indicates that the DDNAME SIRTUNED has an invalid format. A SIRTUNED with invalid format could not have been created by *SirTune* so the SIRTUNED DD card or FILEDEF statement must be pointing to an incorrect dataset.

TUNR0005 Parameter missing for *cmd* command.

This indicates that a parameter was missing for a statement in SIRTUNEI. The line with the invalid statement is echoed before this message. SIRTUNEI must be corrected to produce any reports. The correct format of the statement can be found in "Configuring the Report Generator" on page 37.

TUNR0006 Invalid *cmd* parameter - *parm*.

This indicates that an invalid parameter was found for a statement in SIRTUNEI. The line with the invalid parameter is echoed before this message. SIRTUNEI must be corrected to produce any reports. The correct format of the statement can be found in "Configuring the Report Generator" on page 37.

TUNR0008 **Getmain request failed, RC=*ret_code*. SIRTUNER terminated.**

This indicates that there was not sufficient storage to produce the SIRTUNER reports. Increase the amount of storage allocated to the region with the REGION parameter on the EXEC card under MVS or with the DEFINE STORAGE command under CP. If under CMS/XA or CMS/ESA and 16 megabytes of storage are not sufficient, DEFINE STORAGE for a value greater than 16 megabytes, set the virtual machine to XA mode with the SET MACHINE XA command and run SIRTUNER under this configuration. If none of these options are available, try reducing SIRTUNER's storage requirements with the RESOLUTION command or by reducing the number of reports produced in a single run.

TUNR0009 **REPORT NODEFAULT must be first REPORT command.**

This indicates that the REPORT NODEFAULT was found in SIRTUNEI but it was not the first REPORT statement in SIRTUNEI. Move the REPORT NODEFAULT statement to the top of SIRTUNEI or remove it from SIRTUNEI.

TUNR0010 **SIRTUNED is not in SIRTUNE format.**

This indicates that SIRTUNER has determined that SIRTUNED does not contain valid *SirTune* data. This is most likely caused by coding the incorrect dataset name of the DD card or FILEDEF statement for SIRTUNED.

TUNR0011 **SIRTUNED data is from obsolete SIRTUNE.**

This indicates that SIRTUNER has determined that the data in SIRTUNED was produced by a no longer supported *SirTune* load module. Ensure that the *SirTune* load module is upgraded to the newer release. To produce a report from the already created SIRTUNED dataset, find an older version of the SIRTUNER load module and use it to generate the report.

TUNR0012 **SIRTUNED data is from future version of SIRTUNE.**

This indicates that SIRTUNER has determined that the data in SIRTUNED was produced by release of the *SirTune* load module that is too new for SIRTUNER to process. Ensure that the SIRTUNER load module is upgraded to the newer release and try generating the reports again.

TUNR0013 **No samples in SIRTUNED.**

This indicates that SIRTUNER has determined that no samples were collected into SIRTUNED. This could have occurred if INCLUDE/EXCLUDE statements or a SAMPLE OFF statement for *SirTune* prevented samples from being collected. It could also happen if the run that produced SIRTUNED never made it through *Model 204* initialization.

TUNR0014 ***state* data not collected, cannot produce requested report(s).**

This indicates that reports were requested in SIRTUNEI that require *SirTune* to have collected data for user states for which *SirTune* did not collect data. See "Model 204 States" on page 75 for more information on these requirements. The offending REPORT statement is either a REPORT STATE or a REPORT DISKIO statement. The requested reports cannot be produced with the existing samples dataset identified by SIRTUNED. To produce any reports from the current SIRTUNED, remove the offending REPORT statements from SIRTUNEI. To be able to produce the offending reports for future runs, add the appropriate COLLECT statements to SIRTUNEI for *SirTune*.

TUNR0015 **No samples in requested range.**

This indicates that a RANGE statement was specified in SIRTUNEI that eliminated all samples in SIRTUNED. Correct the RANGE statement or collect data in the appropriate time range with *SirTune*.

TUNR0016 **No non-discarded samples in SIRTUNED.**

This indicates that while samples were found in SIRTUNED they were all discarded as being biased because of the MAXDELAY value. Either change the value of MAXDELAY or correct your system tuning parameters so that the **ONLINE** receives adequate service from the operating system.

TUNR0017 **No non-discarded samples in requested range.**

This indicates that a RANGE statement was specified in SIRTUNEI that limited SIRTUNED to samples that were all discarded as being biased because of the MAXDELAY value. Either change the requested time range, collect data in the appropriate time range with *SirTune*, change the value of MAXDELAY or correct your system tuning parameters so that the **ONLINE** receives adequate service from the operating system.

TUNR0018 **Different Model204 load modules in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were produced with different *Model 204* load modules. Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with identical *Model 204* load modules.

TUNR0019 **Different function tables in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were produced with different *Model 204* load module function tables. Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with identical *Model 204* load module function tables.

TUNR0020 **Different server layouts in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were produced with different *Model 204* server size allocations. Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with identical *Model 204* server size allocations.

TUNR0021 **Different server dataset layouts in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were produced with different *Model 204* server dataset allocations (CCASERVR, CCASERV1, etc.). Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with identical *Model 204* server dataset allocations.

TUNR0022 **MP and non-MP runs in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets one of which was produced using MP (NMPSUBS > 0) and the other not. Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with consistent use or non-use of the MP feature.

TUNR0023 **Different Model204 versions in sample datasets.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were produced with different *Model 204* versions (e.g. 2.2 and 3.2). Ordinarily this is not a problem unless certain reports are requested in SIRTUNEI. The problematic report is indicated by a TUNR0024 message that immediately follows this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report were created with identical *Model 204* versions.

TUNR0024 **Can't produce *rep_name* report.**

This indicates that a DATASET statement was specified in SIRTUNEI that requested combining two sample datasets that were inconsistent in some way. The inconsistency is identified by the immediately preceding error message. The report requiring consistency is indicated by this message. Either eliminate the problematic report or change the DATASET statement to ensure that all datasets to be included in the report have the required consistency.

TUNR0025 **No sample datasets selected.**

This indicates that a DATASET statement was specified in SIRTUNEI that did not specify any sample dataset that was actually in the SIRTUNED concatenation. Either, correct the DATASET statement or add the appropriate sample datasets to the SIRTUNED concatenation.

TUNR0026 ***type* data not collected, cannot produce *rep_name* report.**

This indicates that reports were requested in SIRTUNEI that require SIRTUNE to have collected data for user states for which SIRTUNE did not collect data. See "Model 204 States" on page 75 for more information on these requirements. The requested reports cannot be produced with the existing SIRTUNED. To produce any reports from the current SIRTUNED, remove the offending REPORT statements from SIRTUNEI. To be able to produce the offending reports for future runs, add the appropriate COLLECT statements to SIRTUNEI for *SirTune*.

APPENDIX C Installation

SirTune can be installed from a product tape or can be installed from an object deck downloaded from the Sirius Software web site. Installation from tape is described in "MVS installation" on page 116 and "CMS Installation" on page 118, while installation from the web is described in "Installation from the web". In any case, all *SirTune* distributions come with *all* maintenance pre-applied: up to the time the tape was cut for tape installations, and up to the time the object deck was downloaded for web installations.

As of Version 7.2, reporting is performed by a User Language program that is distributed in the SIRIUS file as part of the *UL/SPF* product family. If your site uses *UL/SPF*, verify both of the following:

1. You have *UL/SPF* Version 7.2 or later installed,
2. File SIRIUS contains the program SIRTUNEREPORT.

If your site is not licensed for *UL/SPF*, you are still authorized (as a user of *SirTune*) to download the SIRIUS file. You can download the SIRIUS file from the Sirius website using the "Download User Language files" link on the Support page (**http://sirius-software.com/support.html**). If you are not a *UL/SPF* customer, you don't need to do the entire *UL/SPF* installation: just download and restore the SIRIUS file, and make it available to your SIRTUNE Report jobs.

If you are a user of a previous version of *SirTune*, see the ***SirTune Release Notes*** for the version you are installing. This document is available on the Sirius Software web site Documentation page (**http://sirius-software.com/maint/manlist**). The Release Notes highlight changes in the new version, and they list any compatibility issues with the previous version.

C.1 Installation from the web

You can download the *SirTune* object files from the Sirius Software web site (**http://sirius-software.com**). The download process requires a userid and password.

1. Click the `Support` navigation link to go to the Customer Service page (**http://sirius-software.com/support.html**), a public page that explains how to get the required userid and password and that also contains links to the various download pages.

2. Click the `Download object files` link to go to a protected page (**https://sirius-software.com/maint/objlist**) that contains a dynamically-generated list of the various Sirius products that you may download.

The page also contains a `Click here` link that causes the page to be re-displayed with detailed download and installation instructions. These instructions supercede any information in this manual.

3. On the `Download object files` page, select the link(s) to the currently available *SirTune* object file(s) for the version you require.

   Clicking a link generates an object file for *SirTune* that is pre-configured for your site, with all maintenance and authorization ZAPs applied. The "Save As" dialog that displays lets you indicate where the object files are to be saved.

   Note: As of *Sirius Mods* version 6.9, the "SirTune" object file is integrated into the "Sirius Mods" object file, so to install the latest version of the *SirTune* components:

   a. Select the "Sirus Mods" link (V6.9 or higher) for the appropriate version of *Model 204*.
   b. Select the most recent "SirTune Report Generator" link.
   c. If you are running under CMS, select the most recent "SirTune Data Logger" link.

   For earlier versions of the *SirTune* components, select the appropriate V1.*x* "SirTune" and "SirTune Report Generator" links, and optionally, for CMS users, the V1.*x* "SirTune Data Logger" link.

   Running an earlier version of *SirTune* with *Sirius Mods* version 6.9 or later is **not** recommended, since older versions of *SirTune* are not upgraded to work with new versions of *Sirius Mods* or *Model 204*.

C.2 MVS installation

SirTune is distributed on a magnetic tape. This magnetic tape has three standard labeled files. These three tape files are:

- SIRIUS.LIB
- SIRIUS.LOAD
- SIRIUS.ULSPF (this file is not used by *SirTune*)

To install *SirTune*, load the files from tape onto disk. The tape contains *SirTune* and any other Sirius products you have ordered. The space allocations are sufficient regardless of how many products you will be loading.

The following JCL performs the load:

```
//SIRTUNEL JOB  (Ø),'Bart
Simpson',MSGCLASS=A,CLASS=C,NOTIFY=BART
//*
//*        Load Sirius products from tape
//*
//IEBCOPY  EXEC PGM=IEBCOPY,REGION=ØM
//T1       DD
UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(1,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.LIB
//T2       DD
UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(2,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.LOAD
//T3       DD
UNIT=TAPE,VOL=SER=SIRIUS,LABEL=(3,SL),DISP=(OLD,PASS),
//             DSN=SIRIUS.ULSPF
//*
//D1       DD   DISP=(,CATLG),DSN=SIRIUS.LIB,UNIT=SYSDA,
//             SPACE=(CYL,(1Ø,2,5))
//D2       DD   DISP=(,CATLG),DSN=SIRIUS.LOAD,UNIT=SYSDA,
//             SPACE=(CYL,(5,Ø,2))
//D3       DD   DISP=(,CATLG),DSN=SIRIUS.ULSPF,UNIT=SYSDA,
//             SPACE=(CYL,(25,Ø,2))
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 COPY I=T1,O=D1
 COPY I=T2,O=D2
 COPY I=T3,O=D3
/*
```

The job card and possibly the output data set names must be changed to conform to local standards.

After loading the two tape files, *SirTune* is ready to use. The first tape file contains some sample jobs, object decks used in building *SirTune*, and a member called ZAP that contains a list of all ZAPs that have been pre-applied to the object decks and load modules. The sample jobs in SIRIUS.LIB are:

LNKTUNE A job that can be used to relink the SIRTUNE load module if necessary. A pre-linked version is shipped in SIRIUS.LOAD.

LNKTUNER A job that can be used to relink the SIRTUNER load module if necessary. A pre-linked version is shipped in SIRIUS.LOAD This module is deprecated as of Version 7.2 because SIRTUNER is no longer supported.

RUNTUNER A sample job that can be used to run SIRTUNER.

All these sample jobs should be modified to conform to local requirements.

As part of the installation process for *SirTune* version 1.5 and earlier: This module is deprecated as of Version 7.2 because SIRTUNER is no longer supported.

- It might be desirable to move the SIRTUNE load module in SIRIUS.LOAD into a library that will allow access from ONLINE and BATCH204 jobs.

- If the ONLINE(s) to be monitored run APF authorized, move SIRTUNE to an APF authorized library so SIRTUNE's monitoring of the ONLINE(s) does not remove the *Model 204* load module's APF authorization. If SIRTUNE's RESTART command is to be available, both SIRTUNE and the *Model 204* load module must be APF authorized.

- If the ONLINE(s) to be monitored run non-swappable, add SIRTUNE to the program properties table (defined in SCHEDxx in SYS1.PARMLIB) as non-swappable, or rename SIRTUNE to something that is already non-swappable.

- If SIRTUNE is to be used to monitor both swappable and non-swappable load modules, an alias should be created for SIRTUNE. Either the name `SIRTUNE` or its alias must then be added to the program properties table as non-swappable. The name (SIRTUNE or its alias) that is added to the program properties table should be used to invoke SIRTUNE for the non-swappable load modules, and the other name should be used to invoke SIRTUNE for swappable load modules.

An object deck called IFINTF is distributed in SIRIUS.LIB that, if linked with one or more IFAM2 programs, will allow the IFCHUNK reports to be produced for any jobs using those IFAM2 programs.

C.3 CMS Installation

SirTune is distributed on a magnetic tape. This magnetic tape has one physical file made up of multiple logical files. The *SirTune* distribution tape is in VMFPLC2 format.

To install *SirTune*:

1. Allocate a minidisk with sufficient space to contain all the Sirius products you ordered. Refer to the tapemap shipped with your installation package for the actual number of blocks required, or allocate a minidisk large enough to hold all Sirius products (about 3,400 4K blocks).

2. Mount the *SirTune* distribution tape on a tape drive, and attach that tape drive to the installing virtual machine as virtual address 181.

3. Access the *SirTune* minidisk as A, then type the following to load all the files required to run *SirTune*:

```
TAPE REW
VMFPLC2 LOAD
```

Included in the loaded files are object decks (with filetype TXTTUNE), load modules (with filetype MODULE), and installation and sample execs (with filetype EXEC).

The execs on the distribution tape are listed below.

Note: It is essential that the CMICONV exec (described below) be run before running SIRTUNE.

BLDTUNE An exec that can be used to relink the SIRTUNE load module, if necessary. A pre-linked version is shipped on the tape. Since SIRTUNE is linked for XA or ESA versions of CMS, run this exec if running a 370 version (Release 5 and earlier) of CMS.

BLDTUNED An exec that can be used to relink the SIRTUNED load module, if necessary. A pre-linked version is shipped on the tape. Since SIRTUNED is linked for XA or ESA versions of CMS, run this exec if running a 370 version (Release 5 and earlier) of CMS.

BLDTUNER An exec that can be used to relink the SIRTUNER load module, if necessary. A pre-linked version is shipped on the tape. Since SIRTUNER is linked for XA or ESA versions of CMS, run this exec if running a 370 version (Release 5 and earlier) of CMS. As SIRTUNER is no longer used as of Version 7.2, this feature is deprecated.

RUNTUNER A sample exec that can be used to run SIRTUNER. As SIRTUNER is no longer used as of Version 7.2, this feature is deprecated.

SIRTUNED A sample exec that can be used as the PROFILE EXEC for the SIRTUNED virtual machine.

SIRTUNEF A sample exec that can be used on the SIRTUNED virtual machine. For more information, see “Collecting Data Under CMS: SIRTUNE” on page 7.

SIRTUNEA A sample exec that can be used on the SIRTUNED virtual machine. For more information, see “Collecting Data Under CMS: SIRTUNE” on page 7.

CMICONV An exec that modifies the load addresses of 'M204CMI TXTLIB' members. It must be run once, before you bring up *SirTune*. The TXTLIB created by this exec should replace the original, or it should be moved to a disk accessed before the original M204CMI TXTLIB.

If SIRTUNE is run against a version 3.2 or later ONLINE without running CMICONV, a member from M204CMI TXTLIB will overlay part of the *Model 204* load module, resulting in a likely 0C4 or 0C1.

4. After loading the tape files:

a. If *SirTune* is version 1.5 or earlier, move SIRTUNE MODULE to a disk that can be accessed by the *Model 204* service machines.

b. Move SIRTUNER MODULE SIRTUNER is not used as of Version 7.2, so this is no longer necessary. to a disk that can be accessed by any user that might wish to produce SIRTUNER reports.

5. Create virtual machine SIRTUNED, and give it access to the appropriate load modules and execs.

   a. SIRTUNED requires the following:

      - PROFILE EXEC (similar to SIRTUNED EXEC from the tape)
      - SIRTUNEA EXEC (similar to SIRTUNEA EXEC from the tape)
      - SIRTUNEF EXEC (similar to SIRTUNEF EXEC from the tape)
      - SIRTUNED MODULE

   b. To run SIRTUNED under the CMS interface (making it possible to write sample data to OS format minidisks), give SIRTUNED access to the M204CMS module shipped with *Model 204*.

   For more details on setting up SIRTUNED, see “Collecting Data Under CMS: SIRTUNE” on page 7.

APPENDIX D Date Processing

SirTune uses dates in the following ways:

- To examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *SirTune* is under a rental or trial agreement
- To display the current date, as returned by the TIME SVC, as page headers in various end-user displays
- To display the date and time of various samples, as returned by the TIME SVC

For headers on pages or rows that occur on printed pages or displayed screens, Sirius Software products generally use a full four-digit year format, although they may display dates with two-digit years in circumstances where the proper century can be inferred from the context.

Index

N

O

P

Q

R

S

T

U

W

X