Getting Started with the IT Service Vision Generic Collector

This document explains how to create a table in your Performance Database (PDB) for use with the Generic Collector facility of IT Service Vision. You would want to use the Generic Collector for data sources which are not supplied with IT Service Vision. An example of such a data source might be a log file generated by an in-house application.

What is the Generic Collector?

IT Service Vision supports data from a variety of collectors, depending upon the platform. On the Open Systems platforms (AIX, Sun Solaris, and HP-UX) data from the following collectors is supported by default:

- NetView 6000
- Unix accounting
- Cabletron Spectrum
- Concord Trakker
- Landmark Probex
- HP PCS/Measureware
- Sun NetManager
- HP OpenView

On the MVS platform, MXG software is used to process data from the following collectors by default:

- SMF-style records
- IMS log
- DCOLLECT data
- DFSMSrmm using the EDGHSKP IBM utility
- Landmark's The Monitor for CICS
- Landmark's The Monitor for DB2
- EREP (SYS1.LOGREC)
- Boole and Babbage's IMS Measurement Facility (IMF)

IT Service Vision includes table and variable definitions for the metrics (data) collected by these collectors. Support for additional collectors may be added in future releases of IT Service Vision.

However you may have a data source (or potential data source) that you wish to process, reduce, and report upon in your PDB. By means of the Generic Collector, you can create your own table and variable definitions, and process this data into the PDB. Almost any time-based data source can be used with the generic collector.

Prerequisites

Before creating a table and processing data using the Generic Collector, you should already have the following experience:

1. Read the Overview chapter (Chapter 1) of the *IT Service Vision Solution: Getting Started* documentation for your platform.
2. Read the chapter on the %CPDDUTL utility macro in the *SAS/CPE Software: Reference, Version 2, Second Edition*, and have some familiarity with using CPDDUTL control statements to create and modify table and variable definitions.
3. In order to accurately define your table and variables correctly you will need documentation for the collector that logs the data. This documentation should be supplied by the collector vendor.
4. You will need write access to a PDB that you want to use for this data.
5. If you want to install the table(s) for your generic collector so that other users at your site may use your table as though it were a supplied table, you will need write access to the IT Service Vision PGMLIB SAS library.
6. You need to be running IT Service Vision Release 1 or greater (SAS Release TS45003 on MVS. SAS Release 6.12 on Open Systems plus the Generic Collector Upgrade pack, available via anonymous ftp at ftp.sas.com.)

Creating a Table for Use with the Generic Collector: an Example

The best way to understand how to create a table for use with the Generic Collector is to actually walk through an example. While each collector is unique, the same basic steps are involved.

For our example, consider an environment in which there are several fax machines, each of which records its activity in an ASCII text log file. Our task will be to take the combined daily log for all of the fax machines, and load that data into the PDB. The fax log for each machine consists of single line message for each fax send attempt and a single line message for each fax receive attempt. Each field in the log file is separated by blanks, and the fields also start in specific columns, as follows:

Columns	Description	Name
1-8	Name of fax machine	MACHINE
10	Send (S) or Receive (R) attempt	TYPE
12-14	Country code	COUNTRY
16-18	Area code	ACODE
20-27	Phone number (xxx-yyyy)	PHNUM
29-30	Pages sent or received (99 page max)	PAGES
32-39	Date fax send/receive began (MM/DD/YY)	DATE
41-48	Time fax send/receive began (HH:MM:SS)	TIME
50-51	Pages re-sent (99 max)	RESENT
53-60	Total (elapsed) time of send/receive (HH:MM:SS)	CONNECT
62	Final Status Code: • Aborted Due to Errors (A) • Receiver was Busy (B) • Completed Successfully (C)	STATUS

Table 1 - Layout of the Fax Log

The **Name** column is simply a list of variable names that we will use later as we work with the data. For an example of the log file, see **Appendix 1 - Example Fax Log.** The **Country Code** field is blank except for international calls.

Overview of the Steps Involved

The process of creating a table for use with the generic collector facility requires the following basic four steps, all of which we will look at in more detail:

1. Staging the Data

In this step we take the raw data and prepare it for use by the %CPPROCES macro. Usually this means writing SAS DATA step code to load the raw data into a SAS data set or view. By doing the staging first, we can use the staged data set as a model for building our table and variable definitions.

2. Creating the Table and Variable Definitions

One component of the PDB is the data dictionary. The data dictionary contains table and variable definitions, also known as ***metadata***. The metadata for our data source must be defined in the PDB by means of one or more table definitions and variable definitions within the table(s). There are many different ways to define a table and variables to the PDB, and we'll look at each of the possible methods.

3. Customizing (Updating) the Table and Variable Definitions

Depending on how the table and variable definitions were created, you may find it necessary to perform additional specifications. This is usually the case when a staged data set is used as a model for building the table and variable definitions. Although tools are provided with IT Service Vision to aid in constructing table and variable definitions, the algorithm used will not always yield exactly what you want. Therefore,

you may want to customize such things as the BY and CLASS variables used, the default age limits for each level of the PDB, what interpretation (type of data) to use for each variable, and what statistics to keep. These customizations should be performed as a series of updates to the generated statements.

4. Setting up for Production Use

Once the table and variables and been defined to the PDB, the necessary processing and reduction jobs must be set up to populate the table with data, using the generic collector.

Now let's look at each of these steps in-depth.

Step 1 - Staging the Data

Staging the data means to transform the raw log file into a SAS data set (or view), which can then be loaded into the PDB. In most cases this is easily accomplished by a simple SAS DATA step, or possibly (on Open Systems) by means of the Character-Delimited variant of the generic collector. For more information about the Character-Delimited Collector, see **Appendix 9 - Using the Character-Delimited Collector**.

Usually only rudimentary SAS DATA step skills are required to stage the data using SAS DATA step code. Before writing the necessary code to stage the data, it is important to understand the raw data, as well as general table-specific and variable-specific concepts and requirements.

Analyzing the Raw Data

You should have some understanding of what the raw data means. For example:

- Does the data break down into distinct types of information, such as incoming and outgoing, or system and user?
- What information exists for providing a date/time stamp?
- Does the data report on a discrete time interval, or does the data record asynchronous (event) activity?
- Can you identify any metrics in the data and how they are to be interpreted?
- Are there any metrics that could be calculated from the raw data, which might be useful for later analysis?
- Can you identify what fields would be useful for categorizing the data? This would typically be the way you would want to categorize and group data for reporting purposes.

In the case of our example, we can answer all of these questions, as follows:

- The data breaks down neatly into faxes that are sent and faxes that are received.
- The DATE and TIME fields provide a means of creating a date-time stamp which we will call TIMESTMP.
- Since each record records the sending or receiving of a fax, the data is EVENT rather than INTERVAL in nature.
- We see that we have counts of pages sent and received, as well as pages re-sent. Additionally we have data for the elapsed time of each fax, as well as its final status.
- We could use the count of pages re-sent and divide it by the total number of pages to arrive at a percent efficiency value.
- Lastly, we categorize the data by MACHINE, TYPE, COUNTRY CODE, AREA CODE, and TIMESTMP.

Table-specific Considerations

IT Service Vision organizes data into tables within the PDB. Typically, each table corresponds to a specific type of record from a collector. Each table consists of variable definitions for the individual fields in the record, as well as attribute information about the table itself.

A table must be defined as either INTERVAL or EVENT in nature. Each table must have a variable called DATETIME, which is the date-time stamp of record. For interval-type tables DATETIME is usually defined as the start of the interval (for ease of reporting), although it doesn't have to be. However, it is best to allow this convention if you are merging multiple data sources for reporting. For event-type tables DATETIME is usually the date-time stamp of when the event occurred. In our example, the TIMESTMP that we created from the DATE and TIME fields will be used as the DATETIME variable. Although it is easier to simply name the timestamp DATETIME in order to avoid confusion, you may find the situation where you are forced to use existing names (for example, if your data already exists in SAS data sets that you only have read access to). Therefore the variable used for DATETIME does not need to be called DATETIME in the staged data.

Additionally, INTERVAL tables must have a variable called DURATION, which contains the elapsed time of the interval that is being recorded. Like DATETIME, this variable does not need to be called DURATION in the staged data. In the situation where the data does not contain a value for DURATION, it may be calculated using the *DIF()* function or similar techniques on the timestamp (DATETIME) values of each record.

Table attribute information includes the descriptive label for the table, how long to keep data at each of the reduction levels (called the *age limits*), and how data is to be sorted (at the detail level) and classified (at the day, week, month, and year levels).

Another important table attribute is the *external name*. Tables in IT Service Vision are limited to 7-character names. The external name of a table, which can be 8 characters, must match the name of the SAS data set (or view) used to stage the data for the table. For user-created tables the table name must begin with the letter U, to prevent name-collisions with supplied table definitions. We will see this later in our example.

Variable-specific Considerations

Variables in IT Service Vision tables also have attributes. Besides having a descriptive label, each variable is assigned a specific *interpretation type*. This interpretation type determines default characteristics such as how a variable is to be displayed, what summary statistics to collect, and how these summary statistics are to be calculated. For more information about interpretation types, see **Appendix 3 - Variable Interpretations and Default Statistics.**

You may tailor which summary statistics will be calculated for each variable. It is important that you understand the interpretation types as implemented by IT Service Vision and know which interpretation type to apply to each of the fields in the raw data.

Another important variable attribute is the *external name*. Variable names in IT Service Vision tables are limited to 7 characters, just as table names are. The variable external name, which can be 8 characters, provides a mapping from the IT Service Vision variable name to the variable in the staged data. We will see this later in our example.

Sample Staging Code

The SAS code shown in **Appendix 2 - Staging Code for the Fax Log** can be used to stage the raw fax log into a SAS data set which we will call WORK.FAXES. The following items should be noted:

1. The staged data set is given a descriptive label, which describes the data set.
2. Each variable is described by an ATTRIB statement, indicating a descriptive label for the variable as well as other attributes.
3. The status code is checked, and if it is invalid, the record is not included in the staged data.
4. The DATE and TIME variables are used to create the TIMESTMP variable. The DATE and TIME variables are not actually kept in the staged data.

5. A new (*derived*) variable (PCTRTRY) is created to monitor the efficiency of the transfer. A derived variable is one that does not exist in the raw data, but can be derived from it. We could instead create a formula variable, but formula variables cannot be used as BY or CLASS variables, nor can statistics be calculated for formula variables. So long as the staging code creates this derived variable the IT Service Vision PDB does not care where it comes from.

Once a staging data set has been built, we can then use it as a model for creating the table and variable definitions for the PDB.

Step 2 - Creating the Table and Variable Definitions

Before data can be processed into the PDB, appropriate table and variable definitions must be constructed to define the data source to the PDB. IT Service Vision allows the flexibility of performing this task in three different ways:

1. Automatically, using the %CPDDUTL utility macro GENERATE SOURCE control statement.
2. Manually, via the IT Service Vision server's interactive graphical user interface.
3. Manually, using the %CPDDUTL utility macro CREATE TABLE and CREATE VARIABLE control statements

Options 2 and 3 are quite labor-intensive, and therefore only recommended for minor alterations or additions. However for more information about these techniques, see **Appendix 4 - Defining Tables and Variables (without Using GENERATE SOURCE).**

Because the data is already staged, the recommended technique for creating the TABLE and VARIABLE definitions is to use the GENERATE SOURCE control statement of the %CPDDUTL utility macro. The GENERATE SOURCE control statement builds the necessary CREATE TABLE and CREATE VARIABLE control statements to create a table in the PDB. It does this by using a model data set. Since we already have a staging data set, we will use this for our model data set, using the DATASET= parameter. For purposes of constructing the data dictionary, the model data set can be created by running the staging code with a null input file.

Other parameters of the GENERATE SOURCE control statement provide information such as the name of the table we want to create (NAME=), whether the table is INTERVAL or EVENT (TYPE=), where to store the generated statements (a SAS catalog SOURCE entry pointed to by ENTRYNAME=), and what variable to use for the DATETIME variable (DATETIME=) or DURATION variable (DURATION=). For more information about the GENERATE SOURCE control statement, see the help entry entitled ***CPDDUTL Generate Source*** in the online help index.

NOTE: The following example makes use of the %CPCAT utility macro to copy in-stream text to a SAS catalog .SOURCE entry. The first invocation of %CPCAT makes use of the SAS language statement CARDS4, to indicate that the end of the data stream is indicated by 4 semicolons, starting in column 1. The second invocation of %CPCAT takes the stream of cards just read, and stores them into the indicated SAS catalog .SOURCE entry. For more information about the %CPCAT utility macro, see the *IT Service Vision Macro Reference Guide*.

For our example the following SAS statements, which can be submitted via the SAS PROGRAM EDITOR window or in batch, generate the control statements for our table, storing the statements in the SAS catalog entry SASUSER.CPDDUTL.FAXES.SOURCE.

```
%cpcat; cards4;
  GENERATE SOURCE NAME=ufaxes
                  TYPE=event
                  DATETIME=timestmp
                  ENTRYNAME='sasuser.cpddutl.faxes.source'
                  DATASET='work.faxes'
                  REPLACE ;
;;;;  /* ;;;; must begin in column 1 to terminate the input stream */
%cpcat(cat=sasuser.cpddutl.generate.source);
%cpddutl(entrynam=sasuser.cpddutl.generate.source);
```

Figure 1 - Invocation of GENERATE SOURCE to Build Example Table and Variable Definitions

To see the CPDDUTL control statements that are generated, see **Appendix 5 - Generated Source Control Statements**.

The generated control statements include attribute information such as the variable interpretation type, the format to use for displaying the data, the default statistics to collect, and the BY and CLASS variables. For more on how GENERATE SOURCE determines what information to use, see **Appendix 8 - Algorithm Used by GENERATE SOURCE**.

Step 3 - Customizing (Updating) the Table and Variable Definitions

Now that GENERATE SOURCE has built the control statements necessary to build our table and variable definitions, we might be tempted to run the generated statements as input to the %CPDDUTL macro. However, it is quite likely that further customization is required. The recommended procedure is as follows:

1. Review the generated control statements, and build a set of CPDDUTL control statements, to override any desired table and/or variable characteristics
2. Run the %CPDDUTL macro, first using the generated control statements as input and then using your update control statements as input.

Review the Generated Control Statements and Build Update Control Statements

Despite the best of algorithms used, however, GENERATE SOURCE is not infallible. Therefore, it is important that you review the generated control statements, verifying that the following characteristics are properly set:

Variable Interpretation Type

As previously discussed the interpretation type chosen for a variable is critical in guiding the summarization and display of the data. You should review the generated control statements, paying attention to the value of the INTERPRET= parameter on each of the CREATE VARIABLE statements. You may find that the algorithm used by GENERATE SOURCE did not specify the interpretation type that is most appropriate. In such a case, you should use the UPDATE VARIABLE control statement to specify the correct value for the INTERPRET= parameter. Valid values for this parameter can be found in **Appendix 3 - Variable Interpretations and Default Statistics**.

Variable Format

The variable format (specified by the FORMAT= parameter of the CREATE VARIABLE control statement) specifies the SAS display format to be used when displaying the data. By default, this is set

according to the interpretation type selected by GENERATE SOURCE. However, if you change the interpretation type you may also want to change the display format. In such a case, you should use the UPDATE VARIABLE control statement to specify the correct value for the FORMAT= parameter. For more information on SAS display formats, see *SAS Language: Reference, Version 6, first edition*.

Statistics

Each variable definition includes a list of requested statistics. You should review this list to make sure that any statistics you wish want will be calculated. Likewise, if statistics have been requested that you would prefer not to keep, you can modify the list to specify that the statistics not be calculated. If no summary statistics are selected for a given reduction level of the PDB (day, week, month, or year) then no data will be reduced for the table at that reduction level.

KEPT=YES/NO

By default, all variables are marked with a kept status of YES so that the PDB detail level keeps data for the variable. However, you may not have any use or need for the data for some variables that may exist in the staged data. In such cases, you can save disk space by marking the variable(s) as KEPT=NO instead of the default KEPT=YES. This allows the definition of the variable to be stored in the data dictionary, without collecting this particular metric.

Formula Variables vs Derived Variables

In our example we have created some new variables that do not exist in the raw data. Such data values are called derived variables, in that they are derived from existing variables. Derived variables have real data in the PDB, just as normal variables do. Therefore, you can use a derived variable as a BY or CLASS variable. You can also request that statistics be calculated for a derived variable.

In contrast, a formula variable can be defined in a table at one or more levels of the PDB. IT Service Vision uses views to access the data in each table. Formula variables are included in the view definition, and thus are calculated dynamically. While this means that formula variables cannot be used as BY or CLASS variables, nor can statistics be calculated for them, they are sometimes preferable because they do not require any disk space for data storage.

BY Variables

The BY variables list designates the sort order for detail-level data for this table. The BY variables list can also be used for duplicate data detection. The BY variables list is also important for reporting purposes. The generated source control statements use the sort order (as known to SAS) of the model data set to determine the BY-variables list for the detail level. This is specified in the UPDATE TABLE control statement at the end of the generated control statements. If the model data set is not sorted, the default BY variables list will consist of the variables DATETIME HOUR SHIFT.

If you want to change the BY variables list you should use the UPDATE TABLE control statement.

Class Variables

The CLASS variables lists designate the sort order for data for this table is for each of the reduction levels (day, week, month, and year) of the PDB. The CLASS variables lists are used to accumulate distinct classes of data, based upon the unique values of the CLASS variables. The CLASS variables lists are also used for reporting purposes. By default, the generated control statements specifies the CLASS variables list for each reduction level identical to the sort order (as known to SAS) of the model data set. However, the CLASS variables lists do not need to match each other nor do they need to match the BY list for the detail level. However, the reduction step (%CPREDUCE) runs optimally if the BY variables list and all CLASS variables lists are the same.

If you want to change the CLASS variables list you should use the UPDATE TABLE control statement.

Age Limits

By default, when IT Service Vision creates a table, it uses the following age limits for determining how much data to store at each PDB level:

PDB Level	Default Age Limit of Data to Keep
DETAIL	10 days
DAY	45 days
WEEK	15 weeks
MONTH	18 months
YEAR	5 years

Table 2 - Default PDB Age Limits

If you prefer to change the age limit for any or all of the levels, you should use the UPDATE TABLE control statement with the AGELIMIT= parameter, specifying the desired age limits.

For an example of some updates for the generated source in our example, see **Appendix 6 - Updates to the Generated Definitions**.

Step 4 - Setting up for Production Use

Now that the table and variables are defined in the PDB and we also have a means of staging the raw data into a SAS data set, we can now process the data. In this section we will discuss the following topics:

- Setting up a batch job to stage, process, and reduce the data for our generic collector
- (Optionally) installing our table and variable definitions so that other users can use them
- Other considerations

Setting up a Batch Job

You probably already have a batch job that processes and reduces data into your PDB each day. This job may also run report definitions. A normal production job for loading data into the PDB has the following three major steps:

1. The %CPSTART macro is invoked, which starts the IT Service Vision server software. This macro points to the PDB that we want to work with and also indicates that we require write access to the PDB.
2. The %CxPROCES macro (either %CSPROCES for Open Systems or %CMPROCES for MVS) is invoked to process the new data into the PDB, populating the detail level of the PDB.
3. The %CPREDUCE macro is invoked to reduce the new data, populating the day, week, month, and year levels of the PDB.

For more information about setting up this daily production job see the *The IT Service Vision Solution: Getting Started* guide for your platform.

To add this generic collector data source to our existing batch job, we must run a separate %CxPROCES macro invocation. The %CPPROCES macro, recommended specifically for the generic collector, can be used instead of the %CMPROCES macro or the %CSPROCES macro. Our example will use the %CPPROCES macro. For more information about the %CPPROCES macro, see the *IT Service Vision Macro Reference Guide*.

The daily job should be modified as follows:

1. The staging code (see **Appendix 2 - Staging Code for the Fax Log**) should be run after the last %CxPROCES macro in the current production job.

2. The %CPPROCES macro should be invoked to process the staged data into the detail level of the PDB. This would be done for our example as follows: `%CPPROCES(UFAXES,GENLIB=WORK);` The GENLIB= parameter indicates which SAS data library contains the staged data for the generic collector table.
3. The existing invocation of %CPREDUCE can be used to reduce data for this table, as well as all other tables in this PDB.

For an example of a daily processing job for our example, see **Appendix 7 - Sample Daily Process and Reduction Job**.

Installing Your Table Definitions in the Master Data Dictionary (Optional)

Other users at your site may want to use this same table in their PDB. We could have every user go through the same steps we went through to create the table and variable definitions. Instead use the INSTALL TABLE control statement of %CPDDUTL to copy the table and variable definitions from the data dictionary in our PDB into the supplied (master) data dictionary. For more information about the INSTALL TABLE statement, see the help entry entitled ***CPDDUTL install table*** in the online help index.

In order for the INSTALL TABLE control statement to work you must have a SAS libref named PGMLIBW established with write access, pointing to the same physical location pointed to by PGMLIB.

For our example, the following SAS statements, which can be submitted via the PROGRAM EDITOR or in batch, will install our table and variable definitions into the master data dictionary. This example assumes that the IT Service Vision server software is in the data set SAS.ITSV.PGMLIB

```
libname pgmlibw 'SAS.ITSV.PGMLIB' ;
%cpcat; cards4;
  INSTALL TABLE NAME=UFAXES REPLACE ;
;;;;  /* ;;;; must begin in column 1 to terminate the input stream */
%cpcat(cat=sasuser.cpddutl.install.source);
%cpddutl(entrynam=sasuser.cpddutl.install.source);
```

Figure 2 - Invocation of INSTALL TABLE to Copy Table and Variable Definitions into the Master Data Dictionary

What to Do When a New Version of IT Service Vision is Installed?

When a new version of IT Service Vision server is installed, the PGMLIB library is replaced. Thus,, after you have installed a new version of IT Service Vision you will have to re-install your table definitions, as shown in **Figure 2 - Invocation of INSTALL TABLE to Copy Table and Variable Definitions into the Master Data Dictionary**.

Other Considerations

As we have seen from our example, the staging code is user-written. As such, you may use your own SAS formats. Also, you may choose to enhance the staging code at some point in the future, or implement support for handling duplicate data. We'll look at all of these issues, and discuss how they should be handled when using a generic collector in a production environment.

User-defined Formats

SAS display formats provide a means to transform the data into meaningful descriptions for reporting purposes. In our example, we could have defined a SAS display format to be used with the fax machine name, which would reflect the office location of the machine. Since this SAS display format would be site-specific and not shipped with the IT Service Vision product, it is your responsibility to maintain the format.

Additionally, IT Service Vision finds SAS display formats by searching first in the catalog DICTLIB.CPFMTS, and then in the catalog SITELIB.CPFMTS. The FMTSEARCH option of the SAS System is used to implement this search order. Therefore, any user-defined SAS display formats should be stored in either SITELIB.CPFMTS (for all users at the site to use) or DICTLIB.CPFMTS (if the format is PDB-specific).

For our example, we could implement a SAS display format for the STATUS variable, to display a meaningful description for each values, as follows:

Value of STATUS variable	Description to display
A	Aborted Due to Errors
B	Receiver Busy
C	Completed Successfully

Table 3 - Display Values for the STATUS Variable

To implement this, we would first construct a SAS display format. For our example, the following SAS statements, which can be submitted via the PROGRAM EDITOR or in batch, will define our SAS display format, storing it in the catalog DICTLIB.CPFMTS. This code must be submitted after %CPSTART has been performed with update access to the PDB.

```
proc format library=dictlib.cpfmts;
  value $STATUS
     'A' = 'Aborted Due to Errors'
     'B' = 'Receiver Busy'
     'C' = 'Completed Successfully'  ;
run;
```

Figure 3 - Defining the SAS Display Format for the STATUS Variable

Note that for any values other than A, B, or C the actual value of the STATUS variable will be displayed instead of the description given by the format.

We must update the definition of the STATUS variable to now use the newly-created format. The following SAS statements, which can be submitted via the PROGRAM EDITOR or in batch, will update the data dictionary accordingly:

```
%cpcat; cards4;
  SET TABLE NAME=UFAXES ;
  UPDATE VARIABLE NAME=STATUS FORMAT=$STATUS ;
;;;;  /* ;;;; must begin in column 1 to terminate the input stream */
%cpcat(cat=sasuser.cpddutl.status.source);
%cpddutl(entrynam=sasuser.cpddutl.status.source);
```

Figure 4 - Updating the Data Dictionary to Associate a SAS Display Format as an Attribute of a Variable

For more information on SAS display formats, see *SAS Language: Reference, Version 6, First Edition*.

What to Do with the Staging Code?

The staging code is user-written and must be managed by the user. It is the responsibility of the user to run the staging code before processing the data for the generic collector. The user is also responsible for any

modifications, changes, and/or fixes to the staging code. Currently IT Service Vision has no facility for managing user-written staging code.

For ease of management of the code you may want to create a SAS macro to execute the staging code. This can be accomplished through the following steps:

1. Determine what to call the macro. For our example we'll call our macro FAXES.
2. Create a member in your SAS autocall library called FAXES. The first line in the member should read as follows: `%MACRO FAXES;`
3. Now, include your staging code as the body of the macro.
4. Type the following as the list line: `%MEND;`

You may now use the macro to run your staging code by simply invoking the macro. This technique is used in **Appendix 7 - Sample Daily Process and Reduction Job**.

Handling Duplicate Data

In order to discuss the handling of duplicate data, it is necessary to distinguish between new data being presented to the detail level of PDB (that is, the staged data), and data already existing in the detail level of the PDB. By default, the IT Service Vision obeys the following conventions with regard to processing data:

1. Prior to being added to the detail level of the PDB, IT Service Vision sorts the new (staged) data using the NODUP option. This removes all duplicates within the staged data.

2. All of the new (staged) data that remains after the sort has occurred is added to the detail level of the PDB, regardless of its age. This allows data to be back-loaded in order to populate the day, week, month, and year reduction levels of the PDB. This data is added to the detail level of the PDB, even if it duplicates existing data already in the PDB. The report at the end of the process run gives a count of the number of records (observations) which have duplicate valued for their BY-variables.

It is possible to alter this default behavior.

In the situation where the staged data should be allowed to contain (intentionally) duplicate observations, process exit PROC095 can be used to override the NODUP option of the PROC SORT, as shown **Figure 5 - Example of Using Process Exit PROC095 to Allow Duplicate Input Data**.

```
%macro proc095;
%if &cptable = UFAXES %then %do;
  ; /* terminate PROC SORT statement */
  *    comment out NODUP option
%end;
%mend;
%proc095
```

Figure 5 - Example of Using Process Exit PROC095 to Allow Duplicate Input Data

For more information about process exits, see the help entry entitled ***ProcessExits 1: Overview*** in the online help index.

By default the generic collector allows data to be processed into the detail level of PDB that duplicates data already in the detail level. The issue of duplicate data being presented to the PDB is essentially an issue of data management, as well as an issue of the collector being used. Several input filtering macros are included with IT Service Vision to aid in dealing with duplicate data. For more information, see the help entry entitled ***CPDUP : Input filtering routines*** in the online help index.

Using the C2RATE and D2RATE Interpretation Type

The C2RATE and D2RATE interpretation types are often the most difficult to understand and use. Consider a situation where a collector regularly collects data every 15 minutes. Each record collected describes the entire 15 minute interval just completed. If a metric exists as some sort of accumulated count, you may want to see it as a rate in the detail level. An example of this would be the number of pages a fax machine sends during the recorded interval. Intuitively, we would consider this metric to be a COUNT. However, since we know that this count will be converted to a rate in the reduction levels (see **STATISTICAL NOTES ON WEIGHTING** in **Appendix 3 - Variable Interpretations and Default Statistics** for more information) we may wish to work with it as a RATE in the detail level as well. Therefore we could choose to assign the C2RATE or D2RATE interpretation type to this metric.

The C2RATE interpretation type should be used in situations where the collector reports a metric in each interval as a counter which has not been reset since the start of the interval. For example, in the first interval the metric would report a value of 50, and in the second interval the metric would report a value of 75. The true value for the count reported by the metric is 50 for the first interval (since the first interval is assumed to start from 0), and 25 for the second interval (since the second interval starts where the first interval left off).

The D2RATE interpretation type should be used in situations where the collector reports a metric in each interval as a counter which has been reset since the start of the interval. Using the same example, the metric recorded in the first interval would report a value of 50, while in the second interval it would report a value of 25.

Using C2RATE and D2RATE requires an extra step with regards to staging the data before running %CPPROCES. The %CPC2RATE macro must be run after the staging code has been run, but before the invocation of %CPPROCES. %CPC2RATE creates as SAS DATA step view to calculate the rates for variables which have the C2RATE or D2RATE interpretation type. An example of using %CPC2RATE is shown below.

```
/* run your staging code here */
%CPC2RATE(tabname=UTABLE,
          in=WORK.TABLE,
          outview=WORK.UFAXES);
%CPPROCES(UTABLE,genlib=WORK);
```

Figure 6 - Example of Using the %CPC2RATE macro to handle C2RATE and D2RATE Interpretation Types

A couple of explanatory notes about this example are necessary:

1. The value for the TABNAME= parameter must specify the name of the table as defined to the PDB.
2. The value for the IN= parameter must specify the SAS data library and data set name where the staged data resides.
3. The value for the OUT= parameter must specify the SAS data library and data view name. The name of the view must match the external name of the table, as defined to the PDB data dictionary.

As part of its processing, %CPC2RATE divides the event count by the value of the DURATION variable in order to determine the rate. As such the C2RATE and D2RATE interpretation types are only meaningful for INTERVAL type tables, due to their use of the DURATION variable.

Advanced Concepts on Staging the Data

At this point our example is complete. The example shown is fairly straightforward with regards to staging the data. In this section we will discuss other data staging situations that you may face. Each of these cases requires deeper experience with writing SAS DATA step code. In each case you would still use GENERATE SOURCE to build the table and variable definitions, as previously discussed.

Data Already in a SAS Data Set

You may find that the data you want to use for your generic collector is already in a SAS data set. If so, you can immediately use GENERATE SOURCE to build your table and variable definitions. You will have to be sure that you have a variable that can serve as your DATETIME variable, and if your are creating an INTERVAL-type table you will also need a variable to serve as your DURATION variable. Again, these need not be named DATETIME and DURATION, respectively; you can designate them on the GENERATE SOURCE statement by using the DATETIME= and DURATION= parameters.

You may find that you need to construct the DATETIME and/or DURATION variables. In such case you may find it easier to create a SAS data step view, in which you build the DATETIME and DURATION variables. You would use the view in place of the staged data set in this case.

Regardless of whether you have the data already in a SAS data set or end up using a SAS data step view, one side effect is that there is no staging code that needs to be run, since the data is essentially already staged.

Data Already in an External DBMS

You may also encounter the situation where the data you wish to stage resides in an external database management system (DBMS). You can use the SAS/ACCESS product to create a SAS/ACCESS view descriptor, which can be used just like a SAS data set or data step view. SAS/ACCESS can read data from the following data sources, on various operating system platforms:

- ADABAS
- Datacom/DB
- DB2
- IDMS
- IMS
- Oracle
- SQL/DS
- Ingres
- RDB
- Sybase
- Informix
- ODBC
- DB2/6000
- DB2/2
- SQL Server
- AS/400
- several popular PC file formats

For more information about SAS/ACCESS, contact your SAS sales representative.

Using a SAS DATA Step VIEW Instead of a SAS Data Set

In situations where you are staging one table from one external data source (as in our example), you may find it advantageous to define a SAS DATA step VIEW instead of a SAS data set. In essence your staging code defines the view. This only works in situations where the entire staging process can be accomplished in a DATA step (that is, no extra steps, such as sorts, are required). The advantage of using a view is that the data are only passed once when they are processed into the PDB. In the normal flow shown in the example, the data are passed once when they are staged and again when they are processed. This technique is useful for large-volume data sources. For more information on the use of SAS data step views, see *SAS Technical Report P-222 Changes and Enhancements to Base SAS Software.*

Reading a Log File or Report (Screen Scraping)

Sometimes the data you want to read are not in a simple log or data file. Instead, the data are actually part of a report. Staging the data essentially becomes an exercise in parsing the report to extract the data values. There is no standard method for doing this. An advanced knowledge of SAS DATA step programming may

be required in such cases. If these skills don't exist at your site, you may want to contact the Professional Services Division of SAS Institute to contract for the necessary services.

Multiple Tables from a Single Input Source

In our example, we had a simple log file that was used to populate a single table. However, we could have also decided to create TWO tables; one for faxes sent, and another for faxes received. The decision on whether to create multiple different tables from a data source depends upon the nature and intended use of the data. In a situation where multiple tables are staged from the same data source, the staging code only needs to be run once and then the %CPPROCES macro can be used to process all of the tables at once.

Character-Delimited Data

Some collectors provide the ability to store the raw data in a character-delimited flat file. Quite often the delimiter is the comma character, spaces, and/or tabs. On the Opens Systems platforms IT Service Vision supports the use of character-delimited data as input for processing a generic collector. Note that in order to use this ability, you must still define the table and variables to the PDB. In the case of character-delimited data, you cannot use GENERATE SOURCE to build the necessary table and variable definitions, since there is no model data set to use.

Support for character-delimited data is only available on the Open Systems platforms, using the %CSPROCES macro. For more information, see **Appendix 9 - Using the Character-Delimited Collector**.

Appendix 1 - Example Fax Log

This is an example of what the fax log looks like. The column rule line is used to indicate the location of the data, as indicated in Error! Reference source not found..

```
Column          1         2         3         4         5         6
Ruler  12345678901234567890123456789012345678901234567890123456789012
       larry    S     502 677-4444  5 10/12/96 13:14:23  0 00:01:45 C
       curly    R     502 677-4444  5 10/13/96 01:01:12  2 00:05:14 C
       moe      S 052 014 752-1234  5 10/14/96 01:01:12  2 00:05:14 C
```

Appendix 2 - Staging Code for the Fax Log

This is the SAS code used to stage the data in the fax log.

```
filename faxlog 'fax.log';

data work.faxes(drop=date time label='Fax Activity');

 attrib machine  label='Machine name'                  length=$8;
 attrib type     label='Fax Type: (S)end or (R)eceive' length=$1;
 attrib country  label='Country Code'                  length=$3;
 attrib acode    label='Area Code'                     length=$3;
 attrib phnum    label='Phone Number'                  length=$8;
 attrib pages    label='Pages Sent or Received';
 attrib timestmp label='Datetime stamp of fax'   format=datetime21.2;
 attrib resent   label='Pages Resent';
 attrib connect  label='Total connect time'      format=time12.2;
 attrib pctrtry  label='Percent retries'         format=percent7.2;
 attrib status   label='Final Status Code'             length=$1;

 infile faxlog;

 input @1  machine          @10 type         @12 country
       @16 acode            @20 phnum        @29 pages
       @32 date : mmddyy8.  @41 time: time8. @50 resent
       @53 connect : time8. @62 status;

 status = upcase(status);
 if status not in('A','B','C') then do;
   put 'Invalid status code found in records ' _n_ ;
   put 'This record will not be included in the staged data';
   put machine= type= country= acode= phnum= pages= date= time=
       resent= connect= status=;
   delete;
 end;

 timestmp=dhms(date,hour(time),minute(time),second(time));

 if pages = 0 then pctrtry = .;
 else pctrtry = resent / pages ;
run;
```

Appendix 3 - Variable Interpretations and Default Statistics

An interpretation type (for example INT, C2RATE, STRING, DATE and so on) is assigned to a variable in IT Service Vision when the variable is created. The interpretation is based upon the characteristics of the data in that variable. The interpretation type defines the characteristics of the data in the variable and determines the default settings for a number of characteristics for the variable (for example, maximum length, format, and so on). In IT Service Vision supplied tables, the interpretation type is already assigned to each of the variables. However, if you add a variable to a supplied table or create a new table of variables, you need to assign interpretation types to your new variables. This appendix describes how to choose the correct interpretation for a variable. This appendix also describes the default settings and statistics that IT Service Vision uses once the interpretation type is assigned.

INTERPRETATION TYPES

Assigning the appropriate interpretation type to a variable is necessary not only for proper formatting for reports but also for proper processing and reduction. Basically, the interpretation type that is assigned provides (implies) the following information:

- The default statistics to keep for the variable and how to calculate them. This is probably the most important implication of the interpretation type.
- The default data type of the variable (character or numeric).
- The default maximum length of the variable.
- The default SAS format to use when displaying the data.

The interpretation types (and their description) supported by IT Service Vision are as follows:

Interpretation Type	Description
AVERAGE	Numeric average
C2RATE	Counter to rate
COUNT	Counter of events
COUNTER	Incremental counter across interval boundaries
DATE	SAS date value (number of days since 1Jan1960)
DATETIME	SAS date time value (seconds since 1Jan1960)
D2RATE	Delta to rate
ENUM	Enumeration number (number of item within a set)
FLOAT	Floating point number
GAUGE	Instantaneous reading
HEXFLAGS	Binary data flags
INT	Integer number
IPADDRESS	Network device address (TCP/IP)
LABEL	SAS label for table or variable
MAXIMUM	Numeric maximum
MINIMUM	Numeric minimum
NAME	String containing one word
NETADDRESS	Network device address
PCTGAUGE	Gauge that shows a percentage
PCTGAUGE100	Gauge that shows a numeric value between 0 and 100
PERCENT	Numeric value between 0 and 1
PERCENT100	Numeric value between 0 and 100
RATE	Events per second
STRING	Printable character string
SUM	Numeric sum
DECADDRESS	Network device address (DECnet)

Interpretation Type	Description
TIME	SAS time value (seconds since midnight)
TIMETICKS	SAS time value (seconds since midnight) originally recorded as hundredths of seconds
UNIXTIME	SAS DATETIME (read as seconds since 1Jan70)
YNFLAG	A single character, either Y or N

Table 4 - Interpretation Types and Descriptions

INTERVAL and EVENT TABLES

IT Service Vision supports two types of tables: INTERVAL and EVENT. An INTERVAL table records performance measures associated with a time interval. For example, an INTERVAL table that records paging subsystem data might include data such as the number of page faults over the life of the interval and the amount of paging space used over the life of the interval. An EVENT table records performance measures associated with an event. For example, an EVENT table that records process termination events might include data such as the time of the process termination, the number of page faults over the life of the process, and the number of I/O operations performed by the process.

Event metrics are recorded only when at event occurs. Interval metrics typically contain data sampled at regular intervals.

DISTINCTIONS AND CLARIFICATIONS

Sometimes it is not clear as to which interpretation type is appropriate for a given metric. Some of the more confusing

DELTA, COUNT, and COUNTER

A DELTA is a variable that represents the difference between two monotonically increasing or decreasing values. The COUNT interpretation is an accumulated count of the number of events, starting at 0 for the interval or event. An everyday example of a COUNT would be a trip-odometer, which gets reset after each trip. In such an example, the miles traveled for the trip are displayed. A COUNTER is an accumulated count of events that does not get reset to 0 (i.e. it spans events or intervals). An everyday example of a COUNTER would be a regular odometer, which only gets reset to 0 when the available digits are used up. Interpretation type COUNTER has very limited usage.

C2RATE and D2RATE

The interpretation type C2RATE is used by the process step in IT Service Vision for Open Systems to convert a COUNTER to a RATE in an INTERVAL table, when the data is processed into the PDB. The interpretation type D2RATE is used by the process step in IT Service Vision for Open to convert a DELTA to a RATE in an INTERVAL table, when the data is processed into the PDB.

INT, COUNT, TIME, and GAUGE

Typically users have trouble determining if a variable should be considered as an INT, a COUNT, or a GAUGE. An INT is a number that stands by itself and for which it makes no sense to perform statistical calculations. An example of an INT might be something like a hardware model number, the number of CPUs installed, or the version/release number of a piece of software. A COUNT is a number that is accumulated, such as the number of page faults for a job. An everyday example of a count would be a trip-odometer reading.

When an average is calculated on a COUNT in an interval table, the resulting value is always a RATE. This is because the only thing that can be used to normalize the interval observations is the DURATION. Hence, count/duration = rate. This is done because there is no way to guarantee that the duration of all intervals will always be the same.

TIME is treated the same as COUNT, because TIME represents an elapsed time, which is nothing other than a count of seconds. A GAUGE is a number that represents an instantaneous reading. An example of a GAUGE would be the number of jobs currently in a given state. An everyday example of a GAUGE would be the reading on a car's speedometer. GAUGE differs from INT, in that statistics are gathered for a GAUGE. GAUGE differs from COUNT in how the average is calculated. In the case of a GAUGE, the average is a simple average, because the instantaneous reading is just that, instantaneous, and does not represent an entire interval.

PERCENT and PERCENT100

A PERCENT is a number between 0 and 1, which represents a proportion. An everyday example of a PERCENT could be a fuel gauge. A PERCENT100 is a number between 0 and 100, which represents the actual mathematical percentage. An example of a variable that could be either PERCENT or PERCENT100 is % CPUBusy. When choosing between PERCENT and PERCENT100, the distinction simply involves whether the data coming into the PDB is between 0 and 1 (PERCENT) or between 0 and 100 (PERCENT100).

DECADDRESS and IPADDRESS

DECADDRESS is intended for DECNet addresses. IPADDRESS is intended for TCP/IP addresses.

DEFAULT STATISTICS

Default statistics are implied by the interpretation type. The default statistics for interval and event tables by interpretation type are shown as codes in the following table. The meaning of the codes follows this table.

Interpretation	Interval Table Statistics	Event Table Statistics
AVERAGE	Aw	csA
C2RATE	Aw	
COUNT	SA	cSA
COUNTER		
DATE		
DATETIME		
DECADDRESS		
D2RATE	Aw	
ENUM		
FLOAT		
GAUGE	csA	csA
HEXFLAGS		
INT		
IPADDRESS		
LABEL		
MAXIMUM	X	X
MINIMUM	N	N
NAME		
NETADDRESS		
PCTGAUGE	csA	csA
PCTGAUGE100	csA	csA
PERCENT	Aw	csA
PERCENT100	Aw	csA
RATE	Aw	csA
STRING		
SUM	S	S

Interpretation	Interval Table Statistics	Event Table Statistics
TIME	SA	cSA
TIMETICKS	SA	cSA
UNIXTIME		
YNFLAG		

Table 5 - Default Statistics Codes for Interpretation Types - Interval and Event Type Tables

The meaning of the statistic codes is as follows:

Code	Statistic Indicated by Code
A	Average
C,c	Count
D	Standard Deviation
h	Sum(DURATION*rate**2)
K	Variance
M	Number Missing
N,n	Minimum
p	Sum(count**2/DURATION)
Q,q	Uncorrected Sum of Squares
S,s	Sum
w	Weighted Sum
X,x	Maximum
Y	Range

Table 6 - Statistical Codes

Lowercase characters indicate that the statistic is hidden from the user. Hidden statistics may be required in order to calculate other requested statistics. For example, if you request the average, IT Service Vision must also keep the sum and the count to calculate the average.

The names of variables in the reduction levels are created by adding these same codes as the final character of the variable name. For example, if you requested sum and minimum for variable CONNECT, its name in the reduction level(s) would be CONNECTS and IFIOCTN, respectively. Variable names that are shorter than 7 characters are padded with underscore (the sum of PAGES would be PAGES__S), and blank is used for Average instead of "A" (the average of PAGES would be PAGES, not PAGES__A).

STATISTICAL NOTES ON WEIGHTING

When calculating an average, there is an implied ``per what'' associated with the average. For data in an EVENT table, the ``per what'' is straightforward: per job, per transaction, etc. Thus the calculation of the average is unweighted.

For data in an INTERVAL table, there is a problem with the obvious first choice of ``per interval.'' For some collectors, the interval can vary dynamically in the data. Others may record in fixed intervals, but the user can change the interval. In both these cases the intervals are not the same, hence it would be misleading to compare or analyze them as if they were. The resolution of this problem depends on the interpretation.

For variables with interpretation type COUNT, TIME, or TIMETICKS calculation is normalized with respect to the DURATION of the interval. By normalizing COUNT and TIME values by the duration of the interval, we arrive at ``per second'' values. This results in rates for COUNT values (for example, page faults per second), and utilizations for TIME data (for example, CPU seconds per second).

For variables with an interpretation type other than COUNT, TIME, or TIMETICKS in an INTERVAL type table, if a weighted average is requested, the calculation of the above statistics is weighted by the duration of the interval.

The calculations for Variance, Standard Deviation, and Coefficient of Variance, if requested, use the same reasoning as these calculations for Average.

DEFAULT TYPE, MAXIMUM LENGTH, AND FORMAT

Default settings for type (character or numeric), maximum length, and SAS display format are implied by the interpretation. This table shows the default settings by interpretation.

Interpretation	Default Type	Length	Display Format
AVERAGE	Num	8	BEST12.2
C2RATE	Num	8	BEST12.2
COUNT	Num	8	BEST12.2
COUNTER	Num	8	BEST12.2
DATE	Num	8	DATE8.
DATETIME	Num	8	DATETIME19.2
DECADDRESS	Char	200	
D2RATE	Num	8	BEST12.2
ENUM	Num	4	5.
FLOAT	Num	8	BEST12.2
GAUGE	Num	6	BEST12.2
HEXFLAGS	Char	200	$hex.
INT	Num	6	BEST12.
IPADDRESS	Char	15	
LABEL	Char	40	
MAXIMUM	Num	8	BEST12.2
MINIMUM	Num	8	BEST12.2
NAME	Char	64	
NETADDRESS	Char	11	
PCTGAUGE	Num	6	PERCENT7.2
PCTGAUGE100	Num	6	PERCENT7.2
PERCENT	Num	6	PERCENT7.2
PERCENT100	Num	6	5.2
RATE	Num	8	BEST12.2
STRING	Char	200	
SUM	Num	8	BEST12.2
TIME	Num	8	TIME12.2
TIMETICKS	Num	8	BEST12.
UNIXTIME	Num	8	DATETIME19.2
YNFLAG	Char	1	$CHAR1.

Table 7 - Data Type, Length, and Format

Note that both PERCENT, PCTGAUGE, and PCTGAUGE100 are displayed using the PERCENT7.2 SAS display format. This display format multiplies a number by 100 before displaying it. Therefore, a PERCENT variable with a value of .2345 would be displayed as 23.45%.

Appendix 4 - Defining Tables and Variables (without Using GENERATE SOURCE)

The recommended technique for defining a new table (and its variables) is to use the GENERATE SOURCE control statement of the CPDDUTL utility macro. However the functionality exists to define tables and variables through the IT Service Vision server's interactive interface, as well as by manually coding the necessary CPDDUTL control statements.

Defining Tables and Variables via the Graphical User Interface

To define a new table using the server's interactive interface, following this path from the main menu:

```
PDB Admin ➜
     Config Active PDB Dictionary ➜
          Actions ➜
               Create Table... ➜
                    enter the name of the table ➜
                         OK
```

At this point you will see a screen that will allow you to enter the table-specific attributes.

To define the variables, follow this path from the table attributes screen you are currently on:

```
Variable List ➜
     Actions ➜
          Create Variable ➜
               enter the name of the variable ➜
                    OK
```

At this point you will see a screen that will allow you to enter the variable-specific attributes.

For help on any of these screens, click on the **Help** button.

Defining Tables and Variables through CPDDUTL Control Statements

For an example of the CPDDUTL control statements necessary to define the table and variables for our example, see **Appendix 5 - Generated Source Control Statements**. For more information about the CPDDUTL utility macro and its control statements, see the *SAS/CPE Software: Reference, Version 6, Second Edition.*

Appendix 5 - Generated Source Control Statements

The following source statements are generated for the example, from Error! Reference source not found..

```
/*-------------------------------------------------------------*
 | Table definition statements generated on 27NOV96:15:12:58.92
 | For PDB table....: UFAXES
 | Table type.......: EVENT
 | Now sorted by....: MACHINE TYPE COUNTRY ACODE TIMESTMP
 | From SAS data set: WORK.FAXES
 | Data set type....: DATA
 | Will be sorted by: MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT
 *-------------------------------------------------------------*/

DELETE TABLE NAME=UFAXES NOERROR ;

CREATE TABLE NAME=UFAXES
  COLLECTOR=GENERIC TOOLNAME=SASDS TYPE=EVENT KEPT=YES
  EXTNAME=FAXES
  LABEL='Fax Activity'
  DESCRIPTION='Fax Activity'
  DETAIL=(agelimit=10 )
  DAY=(agelimit=45  )
  WEEK=(agelimit=15  )
  MONTH=(agelimit=18  )
  YEAR=(agelimit=5  ) ;

  /*-----------------------------------------------------------*
   | Variable definitions for table: UFAXES
   *-----------------------------------------------------------*/

  CREATE VARIABLE NAME=MACHINE EXTNAME=MACHINE
    SUBJECT='N/A' LABEL='Machine name'
    DESCRIPTION='Machine name'
    FORMAT=.
    TYPE=CHARACTER LENGTH=8 INTERPRET=STRING KEPT=YES
    ;

  CREATE VARIABLE NAME=TYPE EXTNAME=TYPE
    SUBJECT='N/A' LABEL='Fax Type: (S)end or (R)eceive'
    DESCRIPTION='Fax Type: (S)end or (R)eceive'
    FORMAT=.
    TYPE=CHARACTER LENGTH=1 INTERPRET=STRING KEPT=YES
    ;

  CREATE VARIABLE NAME=COUNTRY EXTNAME=COUNTRY
```

```
    SUBJECT='N/A' LABEL='Country Code'
    DESCRIPTION='Country Code'
    FORMAT=.
    TYPE=CHARACTER LENGTH=3 INTERPRET=STRING KEPT=YES
    ;

  CREATE VARIABLE NAME=ACODE EXTNAME=ACODE
    SUBJECT='N/A' LABEL='Area Code'
    DESCRIPTION='Area Code'
    FORMAT=.
    TYPE=CHARACTER LENGTH=3 INTERPRET=STRING KEPT=YES
    ;

  CREATE VARIABLE NAME=PHNUM EXTNAME=PHNUM
    SUBJECT='N/A' LABEL='Phone Number'
    DESCRIPTION='Phone Number'
    FORMAT=.
    TYPE=CHARACTER LENGTH=8 INTERPRET=STRING KEPT=YES
    ;

  CREATE VARIABLE NAME=PAGES EXTNAME=PAGES
    SUBJECT='N/A' LABEL='Pages Sent or Received'
    DESCRIPTION='Pages Sent or Received'
    FORMAT=BEST12.
    TYPE=NUMERIC LENGTH=8 INTERPRET=COUNT KEPT=YES
    ISTATS =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    DAY =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    WEEK =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    MONTH =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    YEAR =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    ;
```

```
UPDATE VARIABLE NAME=DATETIME EXTNAME=TIMESTMP
  SUBJECT='N/A' LABEL='Datetime stamp of fax'
  DESCRIPTION='Datetime stamp of fax'
  FORMAT=DATETIME21.2
  ;

CREATE VARIABLE NAME=RESENT EXTNAME=RESENT
  SUBJECT='N/A' LABEL='Pages Resent'
  DESCRIPTION='Pages Resent'
  FORMAT=BEST12.
  TYPE=NUMERIC LENGTH=8 INTERPRET=COUNT KEPT=YES
  ISTATS =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  DAY =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  WEEK =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  MONTH =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  YEAR =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  ;

CREATE VARIABLE NAME=CONNECT EXTNAME=CONNECT
  SUBJECT='N/A' LABEL='Total connect time'
  DESCRIPTION='Total connect time'
  FORMAT=TIME12.2
  TYPE=NUMERIC LENGTH=8 INTERPRET=TIME KEPT=YES
  ISTATS =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
      )
  DAY =(
      NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
      NOVARIANCE NOSTD NONMISS
```

```
        )
    WEEK =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    MONTH =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    YEAR =(
        NOCOUNT SUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    ;

  CREATE VARIABLE NAME=PCTRTRY EXTNAME=PCTRTRY
    SUBJECT='N/A' LABEL='Percent retries'
    DESCRIPTION='Percent retries'
    FORMAT=PERCENT7.2
    TYPE=NUMERIC LENGTH=8 INTERPRET=PERCENT KEPT=YES
    ISTATS =(
        NOCOUNT NOSUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    DAY =(
        NOCOUNT NOSUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    WEEK =(
        NOCOUNT NOSUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    MONTH =(
        NOCOUNT NOSUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    YEAR =(
        NOCOUNT NOSUM AVERAGE NOMAXIMUM NOMINIMUM NOUSS NOCV NORANGE
        NOVARIANCE NOSTD NONMISS
        )
    ;

  CREATE VARIABLE NAME=STATUS EXTNAME=STATUS
    SUBJECT='N/A' LABEL='Final Status Code'
    DESCRIPTION='Final Status Code'
    FORMAT=.
```

```
    TYPE=CHARACTER LENGTH=1 INTERPRET=STRING KEPT=YES
    ;

   CREATE VARIABLE NAME= HOUR
      EXTNAME='HOUR'
      LABEL='Hour of day'
      DESCRIPTION='Hour_of_day'
      KEPT=YES VALIDITY=. INTERPRET=INT TYPE=NUMERIC
      LENGTH=4 FORMAT=2. INFORMAT=. IDNUM=.
      OID= .
      SUBJECT='N/A' ;

    CREATE VARIABLE NAME= SHIFT
      EXTNAME='SHIFT'
      LABEL='Operations shift'
      DESCRIPTION='Operations_shift'
      KEPT=YES VALIDITY=. INTERPRET=STRING TYPE=CHARACTER
      LENGTH=1 FORMAT=. INFORMAT=. IDNUM=.
      OID= .
      SUBJECT='N/A' ;
  /*-----------------------------------------------------------*
   | Default table ordered using DATETIME HOUR and SHIFT.
   *-----------------------------------------------------------*/
  UPDATE TABLE NAME=UFAXES
    DETAIL=(byvars='DATETIME HOUR SHIFT ' )
    DAY=(classvars='DATETIME HOUR SHIFT ' )
    WEEK=(classvars='DATETIME HOUR SHIFT ' )
    MONTH=(classvars='DATETIME HOUR SHIFT ' )
    YEAR=(classvars='DATETIME HOUR SHIFT ' )  ;
```

Appendix 6 - Updates to the Generated Definitions

The following CPDDUTL control statements would need to be executed after the generated control statements shown in **Appendix 5 - Generated Source Control Statements** have been used to initially define the table and variables to the data dictionary. The following control statements will alter the lists of BY and CLASS variables for this table, adding further categorization information.

```
UPDATE TABLE NAME=UFAXES
    DETAIL=(byvars='MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT ' )
    DAY=(classvars='MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT ' )
    WEEK=(classvars='MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT ' )
    MONTH=(classvars='MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT ' )
    YEAR=(classvars='MACHINE TYPE COUNTRY ACODE DATETIME HOUR SHIFT ' )
;
```

Additionally, variable attributes could also be updated as well. A good example of such an update would include updating a variables display format, as shown in **Figure 4 - Updating the Data Dictionary to Associate a SAS Display Format as an Attribute of a Variable**.

For more information about updating the data dictionary, see the *IT Service Vision Macro Reference Guide*.

Appendix 7 - Sample Daily Process and Reduction Job

The following SAS statements are an example of what a daily processing job might look like for the example collector used in this document. This code would typically be submitted nightly by an automated job submission facility (such as cron on Open Systems, or OPC/ESA on MVS).

This job does the following:

1. The %CPSTART macro is executed to start IT Service Vision. For information about %CPSTART, including parameter values, see the *IT Service Vision Macro Reference Guide*.
2. The %FAXES macro is executed to stage the raw data into a SAS data set. See **What to Do with the Staging Code?** for more information
3. The %CPPROCES macro is executed to process the staged data into the detail level of the PDB.
4. The %CPREDUCE macro is executed to reduce the detail level data into the day, week, month, and year levels of the PDB.

You might also want to create a standard set of reports each night, saving the output into SAS catalogs or external files.

The sample job is as follows:

```
%CPSTART(┬..)    /* Start IT Service Vision.  Parameters not shown  */
%FAXES;           /* Stage the raw data into the WORK library        */
%CPPROCES(,UFAXES,GENLIB=WORK);  /* Process staged data into DETAIL */
%CPREDUCE(UFAXES); /* Reduce DETAIL data into DAY, WEEK, MONTH,YEAR */
```

Appendix 8 - Algorithm Used by GENERATE SOURCE

The GENERATE SOURCE control statement of %CPDDUTL uses a model data set to build the table and variable definitions. As the table and variable statements are constructed, certain assumptions are made by the rules embedded in the GENERATE SOURCE logic. By understanding the algorithm used by GENERATE SOURCE, you can decrease the amount of work done to customize the default table and variable characteristics assigned, by building your model data set to take advantage of the algorithm.

The algorithm is used to determine the specific *interpretation type* for each variable. This interpretation type determines how a variable is to be displayed, what summary statistics to collect, and how these summary statistics are to be calculated. For more information about interpretation types, see **Appendix 3 - Variable Interpretations and Default Statistics**.

The determination of the interpretation type is performed in a cascading series of tests, until an assignment is made. Therefore it is important to pay attention to the order in which the heurstics operate, as follows:

First, the algorithm handles the simple case of **character-type data**, as follows:

```
if the variable's SAS display format contains the string 'HEX'
then set the interpretation type to HEXFLAGS
else set the interpretation type to STRING
```

For **numeric-type data**, the algorithm is far more complicated. First the variable's SAS display format is examined to try to set the interpretation type, as follows:

IF the variable format contains the string	THEN set the interpretation type to
'DATE'	DATE
'DATETIME'	DATETIME
'TIME'	TIME
'PERCENT'	PERCENT

If this does not yield a match, the algorithm then begins to examine the variable's label. This is done with special consideration to strings being in the beginning or end of words, hence the use of the blank (' ') as part of the search order. The complete algorithm for this stage is as follows:

IF the variable label contains the string	THEN set the interpretation type to
' PERCENT' '%' 'PERCENT ' ' PCT' 'PCT ' 'PERCENT' 'PCT'	PERCENT100
' RATE' 'RATE ' ' PER SEC' 'PER SEC ' '/SEC ' 'PER SEC'	RATE RATE

IF the variable label contains the string	THEN set the interpretation type to
'/SEC'	
' MAX' 'MAX ' ' HWM' 'HWM ' ' MAXIMUM' 'MAXIMUM '	MAXIMUM
' MIN' 'MIN ' ' LWM' 'LWM ' ' MINIMUM' 'MINIMUM '	MINIMUM
' RATIO' 'RATIO '	PERCENT
' SIZE' 'SIZE ' ' GAUGE' 'GAUGE '	GAUGE
' SUM' 'SUM '	SUM
' DATETIME' 'DATETIME '	DATETIME
' DATE' 'DATE '	DATE
' TIME' 'TIME '	TIME
' AVERAGE' 'AVERAGE ' ' MEAN' 'MEAN ' ' AVG' 'AVG ' ' AVE' 'AVE '	AVERAGE
' TOTAL' 'TOTAL ' ' COUNT' 'COUNT '	TOTAL
'ID' 'VERSION' 'NUMBER' 'STATE' 'TYPE'	INT INT

IF the variable label contains the string	THEN set the interpretation type to
'CLASS' 'PRIORITY' 'PTY'	
'RATE'	RATE
'MAX' 'HWM' 'MAXIMUM'	MAXIMUM
'MIN' 'LWM' 'MINIMUM'	MINIMUM
'RATIO'	PERCENT
'SIZE' 'GAUGE'	GAUGE
'SUM'	SUM
'COUNT'	COUNT
'DATETIME'	DATETIME
'DATE'	DATE
'TIME'	TIME
'AVERAGE' 'MEAN' 'AVG' 'AVE'	AVERAGE
'TOTAL' 'PAGES'	COUNT

If the algorithm has still not assigned an interpretation type, then the interpretation type is set to GAUGE.

Appendix 9 - Using the Character-Delimited Collector

This information is not applicable to the MVS platform.

PURPOSE:

The CHARDELIM option of the GENERIC collector extends IT Service Vision support to virtually any non-binary, flat-file type data logs as long as the data can be reformatted into a standard character-delimited format.

This allows programmers who are not familiar with SAS to use whatever tools they like to format logged systems data and process it into an IT Service Vision PDB. It also allows quick on-site additions of data sources that are not already supported by IT Service Vision.

DATA FORMAT:

- Flat (non-binary) source, maximum record length 4096 bytes.
- The first line must contain a delimiter-separated list of variable names. These names must correspond to the external names of the PDB data dictionary.
- Delimiters must follow the rules for the **DELIMITER=** option of the SAS INFILE statement. See *SAS Language, Reference Version 6* for more information.
- There must be a DATETIME variable. The data for DATETIME must be formatted in a way acceptable to the SAS informat DATETIME19.2. See *SAS Language, Reference Version 6* for more information on the DATETIME19.2 informat.
- If a string variable contains a delimiter character, then the string must be enclosed in double quotes ("). Otherwise enclosing strings in double quotes is optional.
- Two consecutive delimiter characters indicates a missing value.

INVOKING %CSPROCES:

- The parameter **COLLECTR=GENERIC** is required
- The parameter **TOOLNM=CHARDELIM** is required
- The parameter **DELIM=*'delim_chars'*** is optional. If you specify **DELIM=** then *delim_chars* must be enclosed in single quotes. If you do not specify delimiter characters, then a space will be used by default.

EXAMPLE:

To help you get started with processing character-delimited data, the following example will take you through all of the necessary steps to collect vmstat (a standard unix command) data and process it with IT Service Vision.

Several lengthy files are used in this example. These are included with the product in !SASROOT/misc/cpe/. The following provides a list of these files:

File name	File type	Purpose
csvmscol	perl script	Daemon to collect and log vmstat data
csvmsddu	ddutl source	%CPDDUTL source to create vmstat PDB

TASKS:

1. Get csvmscol started collecting data.

The csvmscol script is meant to run continuously in the background on the machine(s) that you wish to monitor. This script logs vmstat data to a log file that you specify. The julian data is appended to the log file name and at midnight a new log file with the new julian date is opened and the old log file is closed. Thus you can process each day's data with IT Service Vision without disturbing the current day's logging activity.

Simply copy !SASROOT/misc/cpe/csvmscol to your working directory and modify as needed. The script header provides more details on using and customizing.

As an example, suppose you wanted to collect vmstat data every 15 minutes on machine "dimes", you would use the following syntax:

```
csvmscol -i 300 -d, -o dimes 2>dimes.err
```

This will gather stats every 300 seconds and log data to dimes.149 (assuming today is julian day 149) using the comma "," as the data delimiter. Stderr is redirected to the file dimes.err. When the script starts up it prints it's pid to stderr so if you want to kill this process you can easily grab it's pid from the dimes.err file.

2. Build a new PDB

To build a new PDB for your vmstat data you will start IT Service Vision and run the %CPDDUTL macro using the control statements in csvmsddu. Submit the following from the SAS program editor:

```
%cpstart(pdb='pdb-vmstat', access=write);
%cpddutl(filename='csvmsddu');
```

This will create a PDB in ./pdb-vmstat/ using the control statements in csvmsddu.

3. Process raw data collected in task 1. into the PDB.

This task will process new raw data and also run reduce on all PDB data. This task assumes that a %cpstart has already been done.

To process your previous days data submit the following from the SAS program editor:

```
%csproces(dimes.149,
        uvmstt,
        collectr=generic,
        toolnm=chardelim,
        delim=',' );
```

To reduce the PDB data submit the following from the SAS program editor:

```
%cpreduce(uvmstt);
```

4. Analyze the data

Once your vmstat data is processed and reduced you are ready to create reports with the reporting tools provided by IT Service Vision

If you browse the data you will notice may variables with missing values. The PDB contains all of the vmstat variables for all three operating systems; AIX, HP-UX and SunOS. So a machine will only have data in those variables that it supports. You may have noticed the following types of types of messages in your SAS LOG during %csprocess:

NOTE: Variable CPUWAIT is uninitialized.

This tells you that CPUWAIT is in your PDB but you did not give any values for it during %csprocess. If you want to avoid these messages you can specify the variable in question as KEPT=NO in the PDB data dictionary. However if you plan to process data from machines supporting all three operating systems in the same PDB you should keep all of the variables.