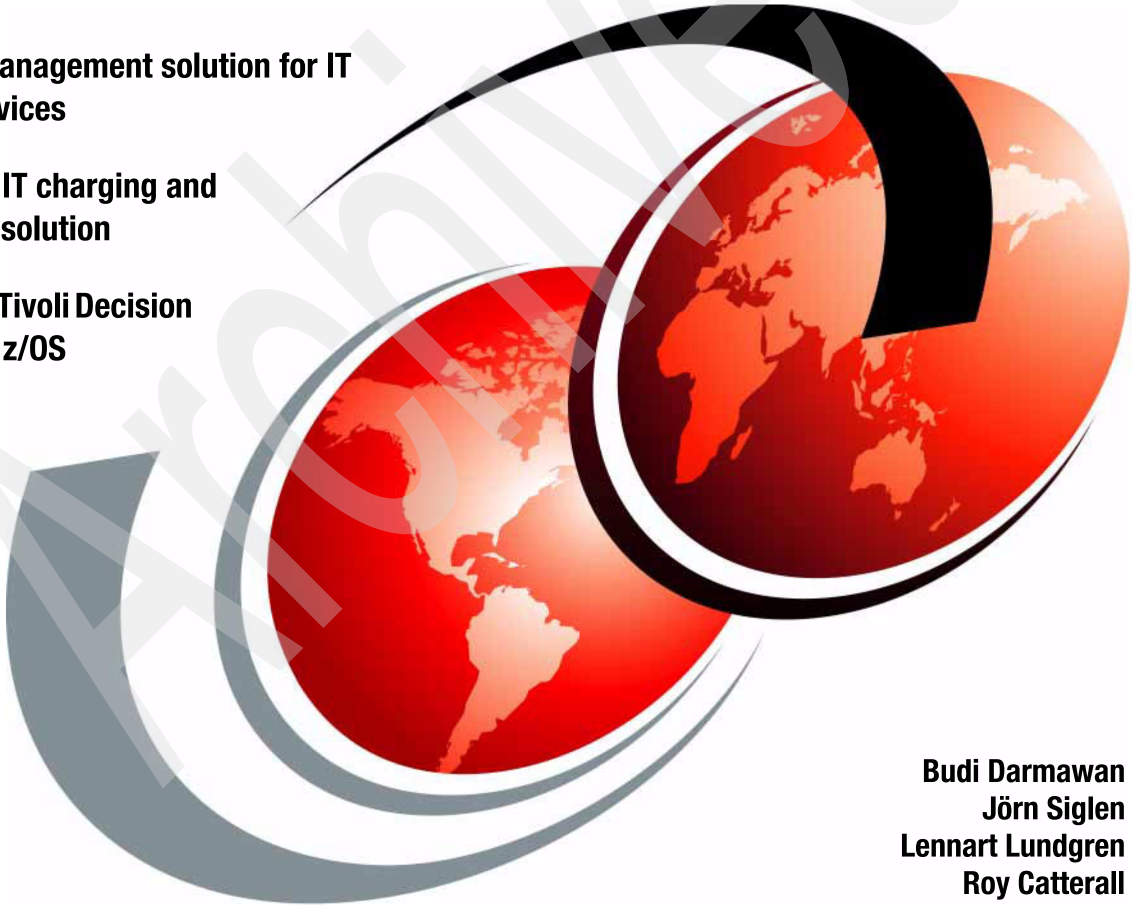
Tivoli. software

IBM

IBM Tivoli Usage and Accounting Manager V7.1 Handbook

- Financial management solution for IT related services
- End-to-end IT charging and accounting solution
- Works with Tivoli Decision Support for z/OS



Budi Darmawan
Jörn Siglen
Lennart Lundgren
Roy Catterall

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Tivoli Usage and Accounting Manager V7.1 Handbook

March 2008

SG24-7404-00

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

First Edition (March 2008)

This edition applies to Version 7, Release 1, Modification 0 of IBM Tivoli Usage and Accounting Manager (product number 5724-O33).

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Redbooks (logo) ®	DB2 Universal Database™	Redbooks®
pSeries®	DB2®	System i™
z/OS®	IBM®	System p™
z/VM®	Lotus Notes®	System z™
AIX®	Lotus®	Tivoli®
Database 2™	MVS™	TotalStorage®
DB2 Connect™	Notes®	WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

IT Infrastructure Library, IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

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

Internet Explorer, Microsoft, OpenType, SQL Server, Visual Basic, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

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

Preface

Financial management of IT resources allows an IT department to be transformed from a cost center to a service provider. One aspect of this is usage accounting, which helps the IT department understand the usage patterns of its customers or users and allows for service charges that reflect that usage. In addition, usage data demonstrates how IT operations can be optimized to increasing efficiency.

Tivoli® Usage and Accounting Manager provides the tools to perform data collection and accounting for IT-related usage from various sources. It even allows the custom integration of data from non-standard format sources. It supports the whole life cycle of financial management from budgeting to usage accounting and billing, to reporting.

This book will help you understand, install, configure, and use the new IBM® Tivoli Usage and Accounting Manager V7.1.

The discussion starts with an overview of Tivoli Usage and Accounting Manager concepts and capabilities along with the structure of the product. The installation and verification of each component is presented in detail. Sample scenarios are executed and explained, including Operating System usage collection, virtual environment collection (VMware ESX server and System p™ partitioning), and Tivoli Decision Support for z/OS® interface.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Budi Darmawan is a Project Leader at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of Tivoli and systems management. Before joining the ITSO 8 years ago, Budi worked in IBM Indonesia as a solution architect and lead implementer. His current interests are Java™ programming, application management, and general systems management.

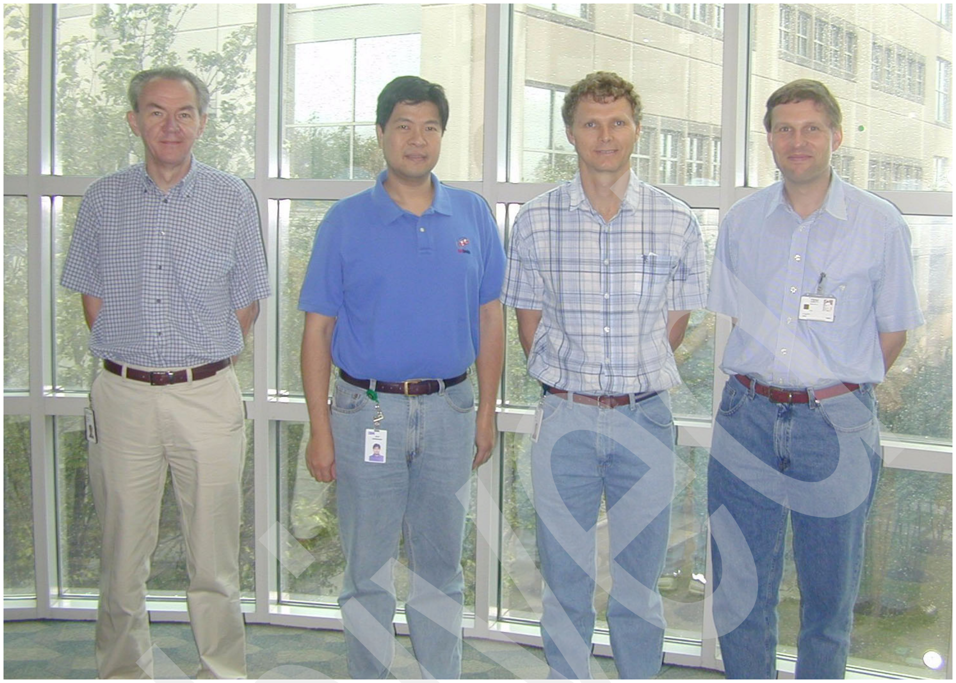


Figure 1 The team: Lennart Lundgren, Budi Darmawan, Roy Catterall, Jörn Siglen

Jörn Siglen is a System Management Architect at IBM Global Services Germany. He has 16 years of experience in the IT field. He holds a degree in Information Technology Engineering from Berufsakademie Stuttgart, Germany. His areas of expertise include AIX® on pSeries® and Tivoli software for monitoring, availability, and storage products.

Lennart Lundgren is an IT Specialist in IBM Software Group, Sweden. He has 30 years of experience in the Systems Management area on mainframe computers. He holds a degree in Computer Sciences from the University of Lund, Sweden. He has worked at IBM for more than 20 years. His areas of expertise include performance and capacity management, z/OS systems programming, and tools development.

Roy Catterall is a Team Leader for Tivoli Decision Support for z/OS in Australia. He has 20 years of experience in the IT field. He holds a degree in Business Studies and Computing Science from the University of Zimbabwe. His main area of expertise is z/OS and he has some programming experience with most other operating systems. He has contributed extensively to the Tivoli Decision Support for z/OS documentation.

Thanks to the following people for their contributions to this project:

Richard M. Conway
International Technical Support Organization, Austin Center

Rodolfo Ambrosetti, Page L. Hite, Greg Howard
IBM Software Group, Tivoli Systems

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks® in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Financial management

This chapter presents an introduction to the IT Infrastructure Library®, highlighting the framework it provides for discussion of IT processes and activities. It addresses some basis concepts related to financial management of certain IT resources. It also includes an overview of the Tivoli Usage and Accounting Manager, the IBM resource accounting product that is the subject of this book. The chapter covers these topics:

- 1.1, "IT Infrastructure Library" on page 2
- 1.2, "Financial management" on page 5
- 1.3, "Tivoli Usage and Accounting Manager" on page 6

1.1 IT Infrastructure Library

Information technology (IT) is crucial to essentially every organization in the current business environment. At the same time IT can be expensive, confusing, and can sometimes appear to not align with overall business objectives. Customers require high quality IT service and security in an environment that also demands compliance with regulatory standards, adherence to specific accounting practices, and often, technological innovation to help the customer maintain a competitive position in their specific industry. All this is set against the backdrop of increasing globalization and rapidly changing technology.

The IT Infrastructure Library (ITIL®) can help address these issues. It is a library of books that document industry accepted best practices for IT service, infrastructure, and application management, designed to help organizations overcome current and future technology challenges. Originally created by the UK Office of Government Commerce (OGC) in 1988, ITIL has evolved as a result of years of experience contributed by major IT organizations and companies, including IBM.

ITIL is an excellent starting point from which to adapt best practices for implementation in any IT environment. Its models show the goals, general activities, inputs and outputs of various IT processes. It helps to address the most common questions asked by IT managers worldwide:

- How do I align IT services with business objectives?
- How do I lower the long-term costs of IT services?
- How do I improve the quality of IT services?

ITIL is currently on its version 3 release. However, the discussion of ITIL in this book is mainly based on ITIL Version 2. Contents of the library are shown schematically in Figure 1-1 on page 3.

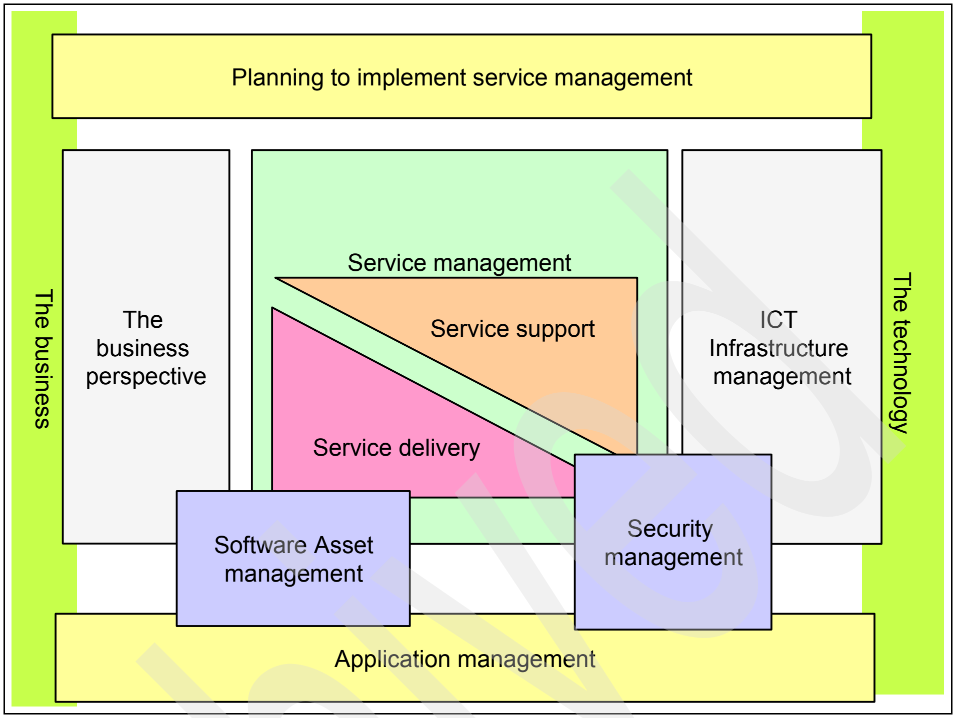


Figure 1-1 The contents of ITIL

The library is organized around the following topics:

- Service support

  Service support focuses on user support, fixing faults in the infrastructure, and managing changes to the infrastructure. The service support addresses the *operational* aspect of IT Service Management.

- Services delivery

  Service delivery focuses on providing services to IT customers. The service delivery topic addresses the *strategic* approach of managing IT Services.

- ICT Infrastructure management

  Information and Communication Technology (ICT) Infrastructure management provides the foundation for service delivery and service support. It provides a stable IT and communications technology infrastructure upon which the services are provided.

- The business perspective

  The business perspective concerns the important aspect of meeting business needs using IT services. This involves understanding the needs of the business and aligning IT objectives to meet those needs.

- Applications management

  Application management describes the application life cycle - from requirements, through design, implementation, testing, deployment, operation, and optimization.

- Security management

  Security management manages a defined level of security for information and IT services.

- Software Asset management

  Software Asset management encompasses the activities involved in acquiring, providing, and maintaining IT assets. This includes everything from obtaining or building an asset until its final retirement from the infrastructure, and many other activities in that life cycle, including rolling it out, operation and optimization, licensing and security compliance, and control of assets.

The most popular books in this library are Service support and Service delivery. These two books together form the Service management discipline. The financial management process is part of Service delivery. Financial management is of a strategic nature; it is used to position IT to perform as a business entity and provides the ability to manage IT as a business.

Configuration information is central to service management; it is generally collected in a database that is typically called Configuration Management Database (CMDB). It should be maintained for and by the operational processes from the service support. The pertinent ITIL-recommended characteristics are as follows:

- Configuration management owns the data in CMDB
- Incident management collects service incident information (questions or disruptions) from the user community and ties it to an entry in CMDB; the entry is called a configuration item (CI). Incident management's goal is to resume service as quickly as possible.
- Problem management provides a structured (long-term) solution of a problem identified, either from incidents or from data or trend analysis from CMDB.
- Change management controls all changes in the IT environment in CMDB. Only approved changes can be performed to a CI; these changes can be physical, logical, or even procedural.

- Release management oversees deployment of CIs into the live or production environment and manages the CIs life cycle.

The configuration data is used in service delivery to build IT services. The following key concepts inform service delivery:

- Service level management manages service level agreements (SLAs) with IT consumers. Service level agreements are the base measurement of IT services that are provided to its consumers.
- Financial management manages the day-to-day IT finances and quantifies IT investment in service improvements. It also generates a balance report of IT budget and accounting.
- Availability management ensures that IT services are available to the business users. It identifies and mitigates risks involved with unavailability due to an IT resource failure.
- Capacity management ensures that IT can provide its services with reasonable performance as dictated by the service level agreement. This requires an adequate capacity of IT resources.
- IT continuity management ensures that IT would continue to function even when a major disruption happens to the business, such as natural disaster.

This book regarding Tivoli Usage and Accounting Manager is closely related to the financial management aspect of ITIL.

1.2 Financial management

The financial management as defined in ITIL involves managing the financial aspect of IT services. As a typical financial discipline, it is concerned with budgeting and accounting of IT services cash flow. With proper financial management, all IT budget can be related to an IT service. The costs of providing IT services can be easily reflected in the task of providing IT services, thus supporting the transformation of IT from a cost center into a business unit that can charge its services for the customers.

The primary goal of financial management is for IT to fully account for money spent and attribute these costs to the IT services delivered. To achieve this goal, financial management must monitor usage and recording costs of IT resources as well as providing an investment business case.

The financial management aspect of IT is more meaningful if IT usage charges are based on business entities instead of on IT entities, and it more accurately reflects the business cost of an IT service. The total CPU time to run a financial application would not be useful for the Chief Financial Officer (CFO); the number of ledger entries processed may be a more meaningful measurement of the financial application usage.

Initially, formulating and calculating these business aspects of the IT services would require a steep learning curve; however, when more information is collected and analyzed, it would be possible to do.

The primary activities for financial management are:

- Budgeting

  Financial management must obtain budget from the enterprise. It administers and controls the cost or expenditure related to the budget.

- Accounting

  Financial management performs financial accounting of IT. It must develop a cost model with its associated cost types, apportion services and calculate costs, and perform Return on Investment (ROI) analysis.

- Charging

  Financial management develops charging policies, identifies charging items, calculates pricing, and performs billing.

Tivoli Usage and Accounting Manager allows the collection of usage data and provides mechanism to input pricing and perform billing. It generates various reports for accounting IT usages and provides financial tools for IT financial modelling.

1.3 Tivoli Usage and Accounting Manager

Tivoli Usage and Accounting Manager is a general purpose tool that does the following tasks:

- Collects resource usage data
- Assigns account codes for each resource
- Provides billing (charging) rates for each unit

Additionally, it provides reports for analysis of the charging environment to ensure that charges are correct and fair. It also comes with a financial modeler feature that allows you to perform rate analysis based on IT expenditure.

IBM Tivoli Usage and Accounting Manager Enterprise Edition V7.1 is a resource accounting product that enables you to track, manage, allocate, and optionally bill end users for IT resources. Tivoli Usage and Accounting Manager Enterprise Edition assists with:

- Usage-based accounting and charge back
- IT cost allocation and analysis
- Application allocation and availability
- Resource utilization reporting
- Easy reporting through a Web interface

Tivoli Usage and Accounting Manager Enterprise Edition consolidates different types of usage metering data into an integrated reporting structure. Tivoli Usage and Accounting Manager Enterprise Edition can then generate reports, invoices, and summary files that show resource consumption and costs for the different functional units within an organization. This information is presented in Web, print, or file formats for easy availability. IBM Tivoli Usage and Accounting Manager Enterprise Edition contains:

- Administration Server, the central component, consisting of the following:
  - Tivoli Usage and Accounting Manager Enterprise Edition Console. This is the Abstract User Interface Markup Language rendering in Integrated Solutions Console (ISC) over the Web Administrator tool.
  - Tivoli Usage and Accounting Manager Engine. This consists of many components, including a batch processing facility called Job Runner that launches and controls the underlying processes that convert raw usage data into usable Tivoli Usage and Accounting Manager Enterprise Edition information. It also contains the main rules engine processing components and other data transformation tools.
  - Generic collection functionality. This consists of the Integrator and the Universal Collection tools, which allow customers to build their own collectors.
  - Tivoli Usage and Accounting Manager Windows® Web Reporting - from Information Internet Services (IIS) under Windows only. This reports directly from the Microsoft® SQL Server®, Oracle®, or DB2® database using Microsoft Reporting Services runtime viewer as the underlying reporting engine and Microsoft IIS as the Web server. This Microsoft Reporting Services viewer must be separately downloaded from Microsoft and installed. It is not supplied with Tivoli Usage and Accounting Manager Enterprise Edition
- Limited Business Intelligence and Reporting Tools (BIRT) reporting directly from the database. If non-Windows reporting is desired, there is a prerequisite that the client will download and install BIRT/IES prior to installation. This reporting can be run from UNIX® or Linux®. While it can also

be run from Windows, the more powerful Tivoli Usage and Accounting Manager Windows Web Reporting is the preferred Windows reporting method.

The Tivoli Usage and Accounting Manager Enterprise Edition - Core Data Collectors, delivered in the same installation as Tivoli Usage and Accounting Manager Enterprise Edition, contain:

- Windows disk usage
- Windows CPU processor usage
- VMware usage collector support
- z/VM®
- AIX Advanced Accounting, including support for Workload Partition, Virtual I/O Server, and any other Advanced Accounting features
- UNIX, Linux, Linux on System z™ operating system
- UNIX, Linux, Linux on System z file system
- System i™ (collects all usage from System i, but the actual collector must be run from Windows)
- Tivoli Decision Support on z/OS extract (similar to the Accounting Workstation Option or IBM Tivoli Usage and Accounting Manager Enterprise Edition for z/OS)
- Generic collection (also known as Universal Collection)
- Miscellaneous and Recurring Adjustment Transaction Maintenance

The Tivoli Usage and Accounting Manager Enterprise Collector Pack (a separate purchasable option) contains additional collectors. In the following lists, a designation of "sample only" means that the collector is not fully documented, it is not globalized or tested, and may not run on all platforms. It is provided as a starting point only, but the sample collectors *will* be supported, via the Level 2/Level 3 support process. A notation of "Windows only" means that the collector or sample only runs under Windows, not under Linux or UNIX. Data is collected about:

- TotalStorage® Productivity Center
- Tivoli Storage Manager (Windows only, other collector may be requested)
- SAP®
- WebSphere® XD
- WebSphere XD HTTP
- Squid (Windows only, sample only)
- Veritas (Windows only, sample only)
- Windows System Resource Monitor (Windows only, sample only)
- Microsoft Reporting Services (Windows only, sample only)
- Evolve (Windows only, sample only)
- Citrix (Windows only, sample only)
- NetWare (Windows only, sample only)
- Oracle

- Oracle Space
- DB2 Usage
- DB2 Space
- Apache Web Server Usage
- FTP transfer usage (Windows only, sample only)
- Lotus® Notes®
- SQL Server (Windows only)
- Microsoft SQL Server database space
- Sybase (Windows only, sample only)
- Apache
- Microsoft IIS
- Microsoft Internet Security and Acceleration (ISA) (Windows only, sample only)
- Microsoft Proxy (Windows only, sample only)
- Netscape Proxy (Windows only, sample only)
- Exchange (Windows only)
- SendMail (Windows only, sample only)
- Windows Print (Windows only)
- NetBackup (Windows only, sample only)
- NetFlow (Windows only, sample only)

New features and capabilities introduced in the latest release, IBM Tivoli Usage and Accounting Manager Enterprise Edition V7.1, are:

- Fully globalized product
- Platform-independent reporting option
- New data collectors
- Improved integration with Tivoli Decision Support for z/OS for mainframe resource accounting
- Web-based administration tool

2

IBM Tivoli Usage and Accounting Manager concepts

This chapter discusses IBM Tivoli Usage and Accounting Manager concepts and architecture. It includes the following sections:

2.1 Tivoli Usage and Accounting Manager components

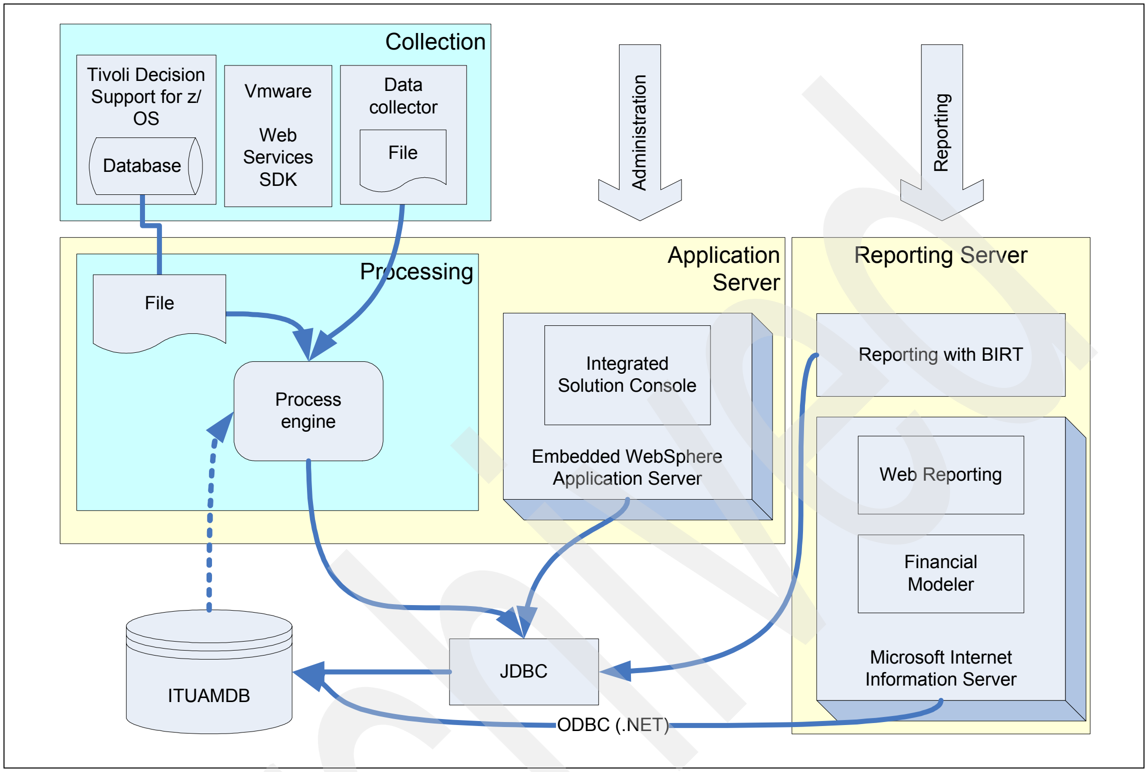


Figure 2-1 Tivoli Usage and Accounting Manager components in use and their dependencies

The main components used by IBM Tivoli Usage and Accounting Manager are shown in Figure 2-1 and described in the following paragraphs.

- Collection

  The collection of metering data is mostly handled by the operating systems and other applications. Tivoli Usage and Accounting Manager data collectors read this data or provide access to the databases where the data is stored. The data collection can be performed from a database table, a file that is converted into Tivoli Usage and Accounting Manager format, or by calling Web Services to collect metrics. We discuss data collection in more detail in 2.3, “Data collection” on page 16.

- Application server

  Tivoli Usage and Accounting Manager application server consists of two primary functions: the administration server and the processing server.

– Administration

  This is performed using the Integrated Solutions Console (ISC). ISC is an application running on top of an embedded WebSphere Application Server. It provides the front end for all administration of the Tivoli Usage and Accounting Manager server. We discuss more on the administration function in 2.2, “Database and administration function” on page 13.

– Gathering and processing of usage and accounting

  The collection of Tivoli Usage and Accounting Manager collector files can be done with a file transfer method or by accessing them directly from a database or Web Services.

  Processing of the data is performed by the Process Engine. It handles all data processing and data loading into the Tivoli Usage and Accounting Manager database. The Java-based JobRunner controls the processing steps. All job descriptions are stored in Extensible Markup Language (XML) files.

  For details about processing see 2.4.4, “Process engine overview” on page 28.

- Database server

  A relational database system is required for storing the administration, metering, and accounting data. Except for reporting (which uses the DB2 .NET interface), the database is accessed using a JDBC™ driver. This driver must be provided for each component that needs access to the database. We discuss the database together with the administration function in 2.2, “Database and administration function” on page 13.

- Reporting server

  All reports are generated from the Tivoli Usage and Accounting Manager database and can be stored on a file system for publishing or distribution. Tivoli Usage and Accounting Manager provides reporting using Microsoft Report Viewer under Microsoft Internet Information Server or using Business Intelligence and Reporting Tools (BIRT). We discuss more about the reporting function in 2.5, “Reporting accounting results” on page 34.

Based on these components, we explain the structure of Tivoli Usage and Accounting Manager in the following sections.

2.2 Database and administration function

The Tivoli Usage and Accounting Manager database is not implemented as part of Tivoli Usage and Accounting Manager installation. Tivoli Usage and

Accounting Manager can have as many database (or data source) definitions as needed and any one of them can be defined as the default. You define these databases to Tivoli Usage and Accounting Manager using the administration server ISC application. ISC performs the following functions:

- Configure access to database and other data sources
- Configure file paths for processing and reporting
- Set up logging level and log file settings
- Initialize or migrate server databases
- Set up access for users and groups, reports, and report groups
- Configure accounting information, such as clients, rate, account code, calendar, and CPU normalization
- Run and monitor jobs using the Job Runner

Most of the administration functions relate to the Tivoli Usage and Accounting Manager database, but some settings are stored in configuration files in the local file system of the administration server.

The administration server processing is shown in Figure 2-2 on page 15.

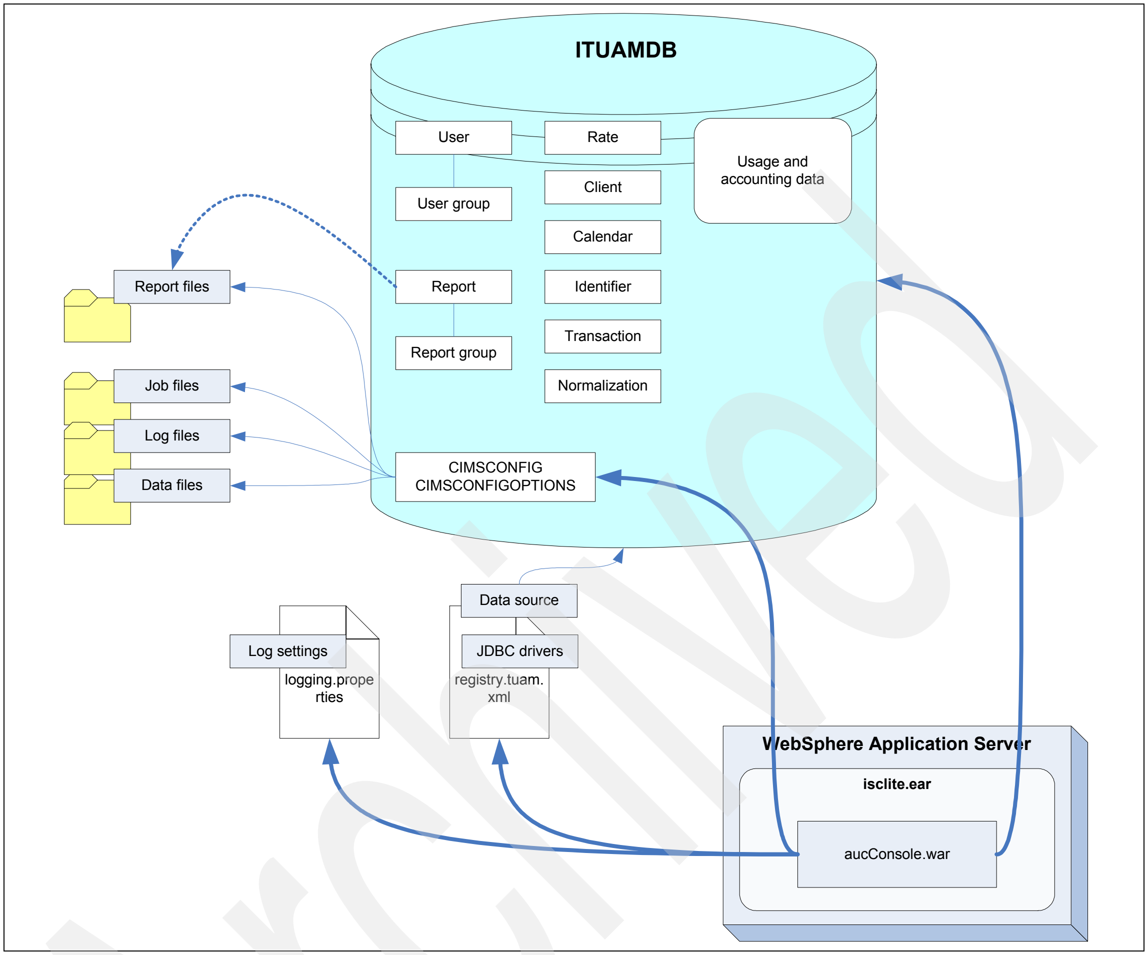


Figure 2-2 Database and administration server

The administration application under the WebSphere Application Server is packaged as a portal-based Web application called aucConsole.war. It provides administrative access to various settings in Tivoli Usage and Accounting Manager as listed in the beginning of this section.

As illustrated in Figure 2-2, the primary system settings reside in:

- CIMSCONFIG and CIMSCONFIGOPTIONS tables in the database. The options in these tables include folder location paths for reports, processing, log files, and data files.
- The JDBC drivers and data sources information is stored in the registry.tuam.xml file. This file is in the local file system.

- The log settings is stored in the logging.properties file.

If you plan to run Tivoli Usage and Accounting Manager processing on multiple machines, or have reporting from multiple machines go into the same database, make sure that they share the same path for either reports or job and log files. This can be achieved if the report files or the job and log files are located in a shared file system (smb, nfs, or other means) that is accessible using the same directory path structure. This is necessary because the folder definition is stored in the database.

2.3 Data collection

Data collection is performed from various data sources. Some of the data sources are:

- Windows disk usage
- Windows CPU processor usage
- VMware usage collector support (collects a small subset of VMware SDK-provided data only)
- z/VM
- AIX Advanced Accounting, including support for Workload Partition, Virtual I/O Server, and any other Advanced Accounting features
- UNIX, Linux, Linux on System z operating system
- UNIX, Linux, Linux on System z file system
- System i (collects all usage from System i, but the actual collector must be run from Windows)
- Tivoli Decision Support on z/OS extract (formerly the Accounting Workstation Option or IBM Tivoli Usage and Accounting Manager Enterprise Edition for z/OS)
- Generic collection (also known as Universal Collection)

Data collection is typically generated into a file and transferred into the processing server for data crunching, analysis, and loading. Some data can also be accessed from a remote system using a JDBC access (such as Tivoli Decision Support for z/OS interface) or Web Services calls (VMware data collection).

2.3.1 The core data collectors

The Tivoli Usage and Accounting Manager core license server includes the following ready to use data collectors:

- AIXAAInput: AIX Advanced Accounting for logically partitioned System p installation that includes support for AIX V5, AIX V6, and the Virtual I/O (VIO) server. (See 6.1, “System p virtualization and AIX Advanced Accounting” on page 160)
- Base UNIX collector: The UNIX collector runs on most UNIX platforms using the build in accounting (acct) features. See 5.2, “AIX data collection” on page 129 for more information.
- CSRInput: Input with Common Source Format (see 2.4.2, “The Common Source Resource format” on page 22). This is typically an output from Tivoli Usage and Accounting Manager data collector or previous processing from Tivoli Usage and Accounting Manager.
- System i: Only available for Tivoli Usage and Accounting Manager on Windows, as a Windows script file, data is collected from i/OS release 5.1.
- TDSz: Extracting data from Tivoli Decision Support for z/OS database (DRLDB). More on this is in Chapter 7, “Processing data from Tivoli Decision Support for z/OS” on page 201.
- Transaction: A transaction is a mechanism to adjust data in Tivoli Usage and Accounting Manager. This collector gets the input from a table within Tivoli Usage and Accounting Manager database, and adds onetime charges and monthly fixed charges to the accounts based on the input from ISC. An example is discussed in 8.2, “Transaction data collector” on page 265.
- z/VM: This collects data from the z/VM environment, including connect time, CPU time, virtual SIOs, virtual cards read, virtual lines printed, virtual cards punched and temporary disk space.
- VMware: This can pull data from either the VMware Virtual Center Server or directly from VMware ESX servers using the VMware SDK Web interface. See 6.2, “Virtualization with VMware” on page 182 for details.
- Windows Disk Data: This program runs on the Windows server every time you want to have a snapshot of disk usage.
- Windows Processor collector: A service that is installed and run in Windows environment to collect data on processor usage.
- Universal data collector is a converter function to convert data into CSR or CSR+ format. The input can be from:

DATABASE	databases providing SQL interface
DELIMITED	delimited files, like comma separated values (csv))
FIXEDFIELD	fixed field files

See also 3.4.1, “The Job Runner integrator collector” on page 56 for more details.

2.3.2 The Enterprise Collector Pack collectors

All additional application-specific collectors are bundled in this package, which has to be installed on top of the base Tivoli Usage and Accounting Manager application server.

- ApacheCommonLogFormat: Apache HTTP server common log collection for analyzing Web page hit count.
- DB2: Uses the event log and data file to get usage data from SQL server.
- DBSpace: Collects the size of a Microsoft SQL or Sybase database only.
- Lotus Notes: Gathers data directly from Notes database files log.nsf, loga4.nsf and catalog.nsf, such as NotesDatabaseSizeInput, NotesEmailInput, NotesUsageInput.
- Microsoft Exchange: Based on the different logs for the Exchange server, usage data and mailbox size are collected.
- Microsoft Internet Information Services (IIS): The W3C Extended Log from IIS can be retrieved for processing.
- Microsoft SQL server: Uses the trace log and direct database access to get usage data from SQL server.
- Oracle: Uses the event log and direct database access to get usage data from Oracle server.
- SAP: SAP Transaction Profile report (ST03N) is used for collecting from SAP. ST03N is a specific transaction in SAP that provides performance and workload analysis data.
- Tivoli Storage Manager: Uses Tivoli Storage Manager ODBC calls (Windows only, but other versions can be requested).
- TotalStorage Productivity Center (TPC): A flexible data collector to collect any data from the TPC log files.
- WebSphere: A variety of WebSphere usage metrics can be collected and processed.
- Windows Event Log data collector for print: Gets usage data from a Windows print server extracted from the event log.

2.4 Processing server

The processing function of Tivoli Usage and Accounting Manager is a very versatile batch job processing function within the Job Runner. The Job Runner models a multi sectioned job. In this section, we discuss:

- 2.4.1, “Generic processing overview” on page 19
- 2.4.2, “The Common Source Resource format” on page 22
- 2.4.3, “Account code and rate” on page 24
- 2.4.4, “Process engine overview” on page 28

2.4.1 Generic processing overview

The data processing in Tivoli Usage and Accounting Manager is similar for all data sources. Figure 2-3 on page 20 shows the general processing steps for data handling with IBM Tivoli Usage and Accounting Manager. The order or mix of the steps may be different, depending on the collectors used.

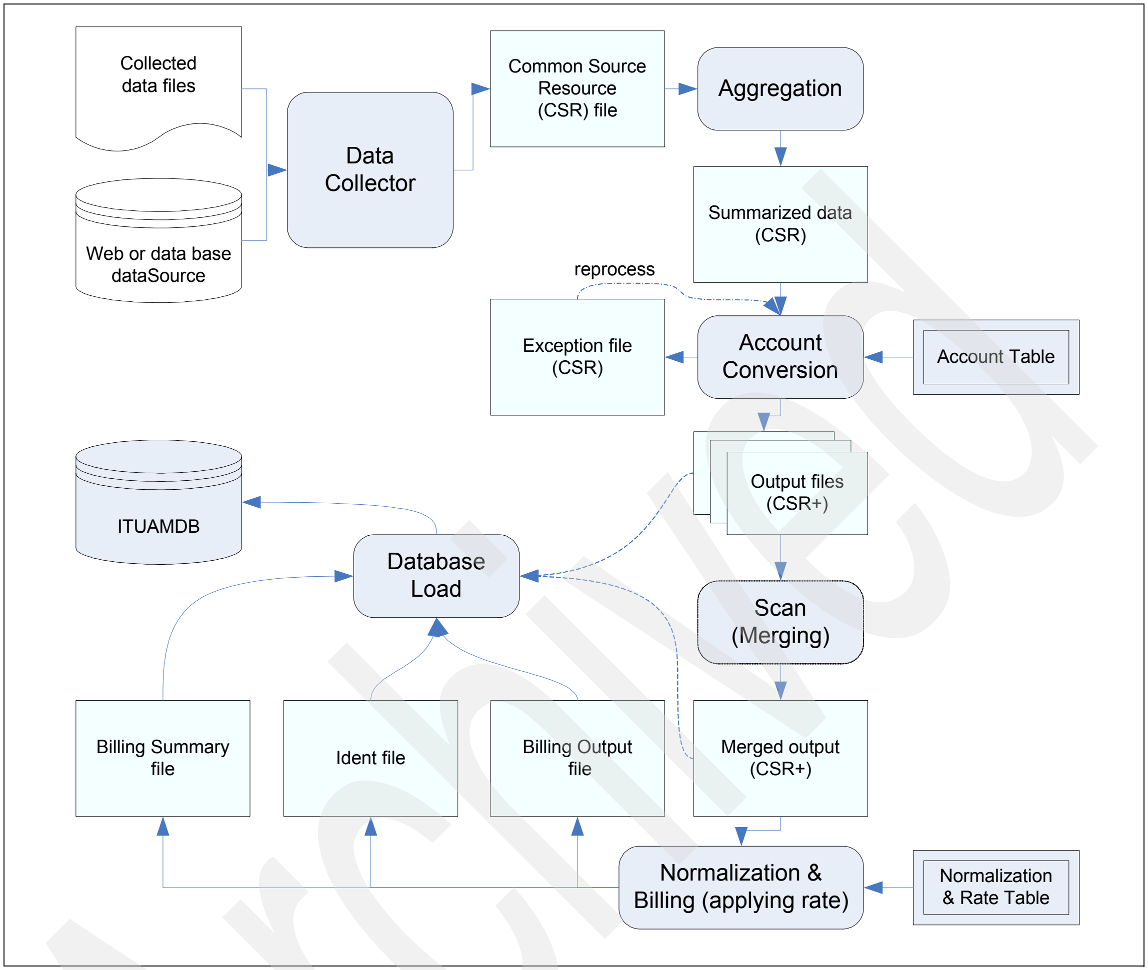


Figure 2-3 Generic process overview, including common steps

The process steps in Figure 2-3 are:

1. Many systems already have a resource usage collection function and Tivoli Usage and Accounting Manager will use this data for further processing. The main processing in Tivoli Usage and Accounting Manager is based on Common Source Resource (CSR) format. The initial processing step converts the existing data (SQL table, delimited file, or others) into CSR format prior to Tivoli Usage and Accounting Manager processing.
   a. If the metering data is collected in files, these files are transferred to the application server and converted to CSR format if needed. Some converters may also include pre aggregation.

   b. If the metering data can be accessed on a database or web page, the data extract made by Tivoli Usage and Accounting Manager will be directly into CSR format.

   The Tivoli Usage and Accounting Manager Integrator can include CSR conversion, aggregation, account code conversion, and sort in one step, thereby producing only one output file.

2. CSR data is aggregated mostly on a daily basis. Aggregation means summarizing the data based on given identifiers. It groups rows of data based on the identifier values; all the resource fields are added up as the aggregation method.
3. Account conversion matches the metering data to the account code structure (see 2.4.3, "Account code and rate" on page 24) and all records that do not fit are put in an exception file; this exception file might be reprocessed later after some intervention.
4. CSR or CSR+ files of the same type can be scanned into one file at any time during processing.
5. Normalization of CPU values and multiplying by the rate code is the next step. The selected rate table is used for calculating the money value. If the rate is based on the type of CPU, recalculation based on the Normalization table is done in addition.

   Data is summarized on financial and organization levels, which provides the billing files: billing detail, billing summary, and identifier list.

6. Loading all output data into the Tivoli Usage and Accounting Manager DB completes the processing. There is an automatic duplicate detection that prevents duplicate data loading.

> **Note:** We recommend creating CSR+ records as input for the billing step, or alternatively using the Integrator Sort on the account code. The number of billing summary rows in the database can be reduced on a CSR file sorted by the account code. CSR+ data is automatically sorted by the bill process.

2.4.2 The Common Source Resource format

Tivoli Usage and Accounting Manager uses two file formats called Common Source Resource (CSR) and Common Source Resource plus (CSR+). The CSR+ is enhanced by a static header that includes the account code for sorting purposes. CSR+ and CSR files are comma separated files, in which each record has these three sections:

- Header

  The header of the record contains the following:

CSR Plus Header	CSR+ records only start with:
"CSR+	constant
headerstartdate	Usage start date
headerenddate	Usage end date
headeraccountcodelength	Length of the Account code (three digits)
headeraccountcode	Account Code
"	constant
headerrectype	Record type or source
headerstartdate	Usage start date
headerenddate	Usage end date
headerstarttime	Usage start time
headerendtime	Usage end time
headershiftcode	Shift code

  The header information is used to identify the applicability of the record to a certain billing period and type.

  > **Tip:** All **header%** variables can be used with the Integrator identifier functions.

  A sample header segment for CSR is:

  ```
  UNIXSPCK,20071016,20071016,00:00:00,23:59:59,1
  ```

  A sample header for CSR+ starts with:

  ```
  "CSR+2007101620071016009AIX 0Test",UNIXSPCK,20071016,..
  ```

- Identifiers segment

  The identifiers segment lists the resource identifier. These identifiers are used to distinguish resources from each other before mapping them to an account code. The account code itself is considered an identifier. The structure of this segment is:

  number of identifiers, identifier name, identifier value, ...

  An example of an identifier segment with 3 identifiers is:

  ```
  3,SYSTEM_ID,"lpar04",Account_Code,"AIX 1TEST  lpar04", USERNAME,"root"
  ```

- Resources segment

  The resources segment lists the resource metrics. These metrics are used to meter the usage information for the resource. The resource metric is structured as:

  number of resources, resource metric name, resource metric value, ...

  An example resources segment with 3 metrics is:

  `3,LLG102,17.471,LLG107,6.914,LLG108,3`

Example 2-1 shows the data from two AIX LPARs on two different systems.

Example 2-1 CSR file for AIX Advanced Accounting data

```
AATRID10,20071030,20071030,01:10:03,01:10:03,1,2,SYSTEM_ID,"02101F170",
Account_Code,"AIX 1TEST  lpar04",1,AAID1002,0.016
AATRID10,20071030,20071030,01:15:03,01:15:03,1,2,SYSTEM_ID,"02101F170",
Account_Code,"AIX 1TEST  lpar04",1,AAID1002,0.004
AATRID4,20071030,20071030,02:30:07,02:30:07,1,2,SYSTEM_ID,"02101F25F",A
ccount_Code,"AIX OSAP   ohm01",2,AAID0402,120,AAID0407,2048
```

In Example 2-2 we find the data from two VMware ESX servers (SYSTEM_ID) and three VMware guests (Instance) collected using a single VirtualCenter Server (Feed).

Example 2-2 CSR file for VMWare processing

```
VMWARE,20071017,20071017,00:00:00,23:59:59,1,5,HostName,"host-19",Insta
nce,"vm-33",Feed,"ITSC_VC",Account_Code,"WIN 1ESX",SYSTEM_ID,"srv079.it
sc.austin.ibm.com",1,VMCPUUSE,10756036
VMWARE,20071017,20071017,00:00:00,23:59:59,1,5,HostName,"host-19",Insta
nce,"vm-41",Feed,"ITSC_VC",Account_Code,"WIN 4ESX",SYSTEM_ID,"srv079.it
sc.austin.ibm.com",1,VMCPUUSE,10688008
VMWARE,20071017,20071017,00:00:00,23:59:59,1,5,HostName,"host-8",Instan
ce,"vm-31",Feed,"ITSC_VC",Account_Code,"WIN 0ESX",SYSTEM_ID,"srv106.its
c.austin.ibm.com",1,VMCPUUSE,637429
```

The Tivoli Usage and Accounting Manager defines some reserved identifiers that are used for special processing. Those identifiers are:

Account_Code Will be matched with the Account Code Structure and used for Rate Table selection and Reporting Aggregation

SYSTEM_ID Used for reading the factor from the Normalization Table during CPU normalization

WORK_ID Identifies a subsystem for CPU normalization such as TSO, JES2, or other subsystems (even non z/OS related ones). This field is optional.

Feed Identifies and defines a subfolder within the process folder for data transfer

2.4.3 Account code and rate

Account code is the primary identifier that signifies who should be billed for the specified system usage. The account code structure has to be defined early on, before you perform any data collection and processing. All the data items will be labelled by the account code; therefore, it would be very hard to change the structure. This section explains the usage of the account code within Tivoli Usage and Accounting Manager and should help you to define the account code structure according to your needs.

Account code is a string with fixed width field that defines the hierarchy of the accounting breakdown. The fields could be used to split the account string for charging different organizational entities. Figure 2-4 shows a sample account code and its relationship to charging rate.

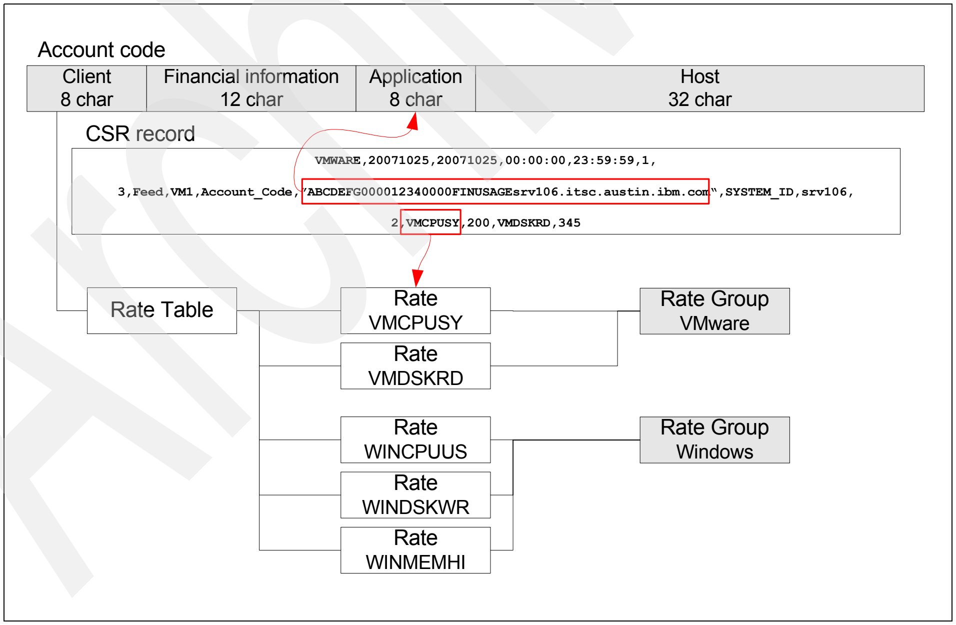


Figure 2-4 Sample Account Code with four parts and the Rate Code relationship

The first part of the Account Code is the Client, representing the top level of your organization. The other parts are hierarchical information for aggregating the data during reporting. All parts of the Account Code are used to search the Clients table to get a rate table. The lookup is performed based on each level of the Account Code hierarchy level. If no match is found, it will use the STANDARD rate table. You can set up a specific Rate Table for any account if needed. The Rate in the specific rate table is matched to the resource name in the resources segment of the CSR file to get the appropriate rate information.

Rates are also organized in rate groups. The rate group allows you to report summary usage based on rate groups. Each rate has definitions about the format, type, conversion factor, and money value for all shifts.

> **Restriction:** Defining a new Rate Group using the ISC Rate menu is limited to eight characters only. Using the ISC Rate Group menu you can rename it later or create longer names, as the examples shipped with Tivoli Usage and Accounting Manager are using.

If a Rate has the type CPU, the normalization will be done for this value during billing based on the identifiers SYSTEM_ID or WORK_ID, or both.

The default account code structure looks like Figure 2-5. This can be maintained using Integrated Solutions Console (ISC) menu **Usage and Accounting Manager** → **System Maintenance** → **Account Code Structure**.

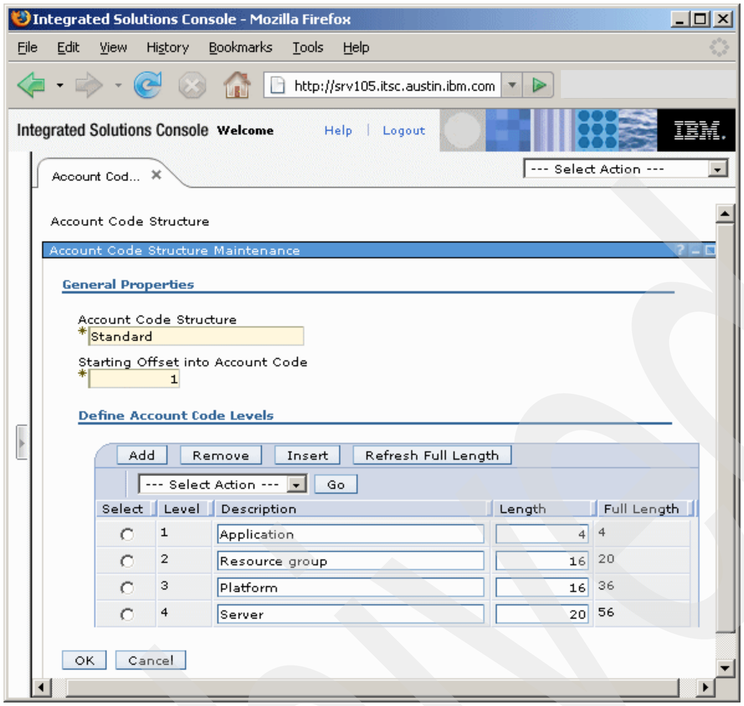


Figure 2-5 Default Tivoli Usage and Accounting Manager Account Code Structure

Best practice recommendations for Account Code Structure

In Table 2-1 we use the department name as the top level and Client that is assigned to a Rate Table based on this department name. There are one or more department numbers possible for each department and the hosts are grouped by an application perspective.

Table 2-1 Account Code for a department organization

Description	Length
Department (short name)	8
Department number	6
Application	8
Host	32

Table 2-2 on page 27 shows an account code structure for a multi customer environment. The customer name is the top level and Client for assigning the Rate Tables on the customer level. We define two additional levels in case the customer needs separated bills or rates for his projects, or and split off on follow-up contracts. The rest stays the same as before.

Table 2-2 Account Code for a multi customer environment

Description	Length
Customer (short name)	8
Master contract number	12
Service contract number	12
Application	8
Host	32

Tip: Changing the account code structure makes already processed data invalid because the values are based on different account code fields. You must plan the structure carefully *before* you start processing data.

Normalization of CPU values

To account for the different types of processors, the metering values for different hardware can be normalized to be comparable.

The settings we need for CPU normalization are:

Rate definition The rate definition must include the check mark for the CPU value.

Identifier An identifier of the name SYSTEM_ID must exist and optionally the identifier WORK_ID; the WORK_ID is used as a prefix to the SYSTEM_ID for lookup of the variable in the billing process.

controlCard The billing step will need the parameter `controlCard="NORMALIZE CPU VALUES"`

Normalization table We have to define all possible values for the identifier SYSTEM_ID or WORK_ID + SYSTEM_ID in the database using ISC menu **Usage and Accounting Manager** → **System Maintenance** → **CPU Normalization**.

Figure 2-6 presents an overview on the normalization function.

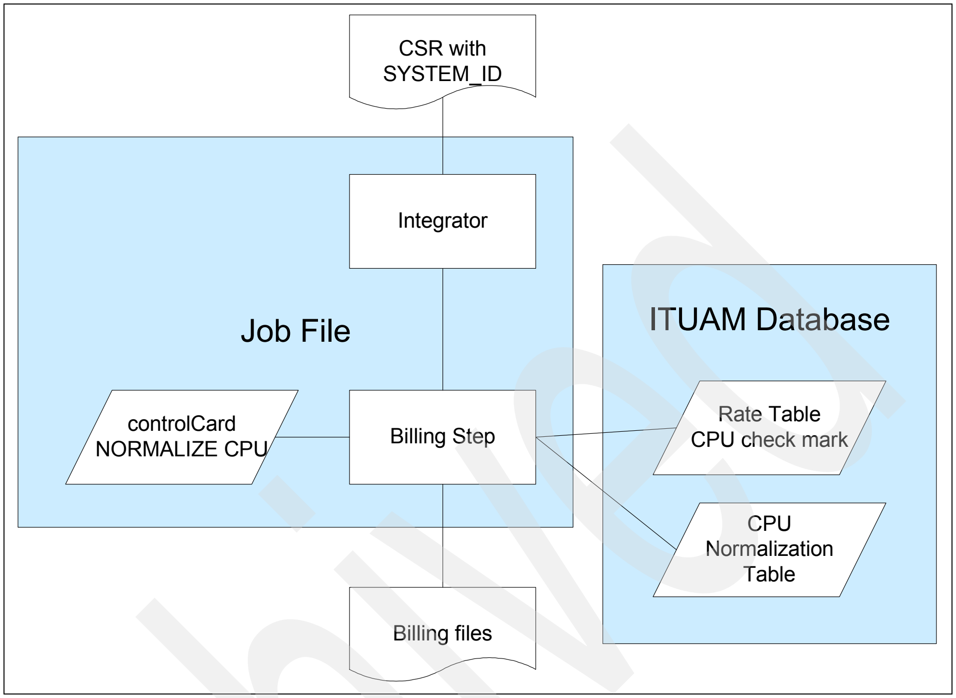


Figure 2-6 Normalization process overview

> **Important:** We recommend performing normalization only within the same platform. There is no good way to generate a comparable CPU value for different processor architectures.

2.4.4 Process engine overview

The process engine of Tivoli Usage and Accounting Manager handles all data processing and loading to the Tivoli Usage and Accounting Manager database. The process engine is controlled by execution of a job in Job Runner. Jobs are described in XML job files. These XML files are provided as samples by Tivoli Usage and Accounting Manager.

Details about Job Runner are discussed in the following sections:

- "Job Runner graphical user interface" on page 29
- "XML structure of Job Runner" on page 31
- "Scheduling Job Runner job files" on page 33

Job Runner graphical user interface

Figure 2-7 shows the graphical user interface (GUI) for job handling. The menu accessed by selecting **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files** includes validation of the XML syntax, running of the job, and can be used for small changes to an existing job. For editing and creating larger job files we recommend using a specialized XML editor.

> **Note:** Because the job file must reside in the jobfiles directory in the processing server, the XML editor should have access to this folder or path.

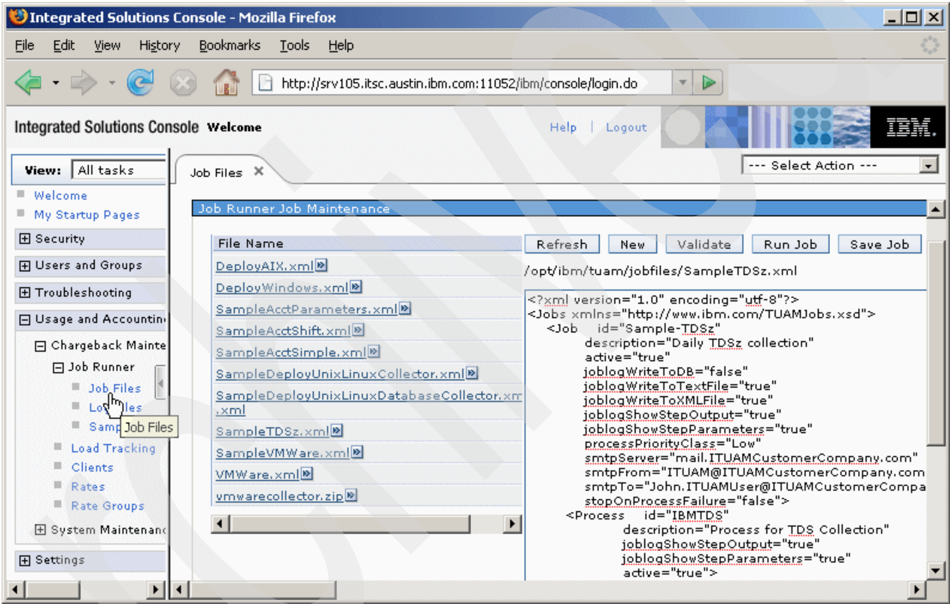


Figure 2-7 The Job Runner Job maintenance

The detailed log files can be viewed using the GUI and selecting **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Log Files** (Figure 2-8).

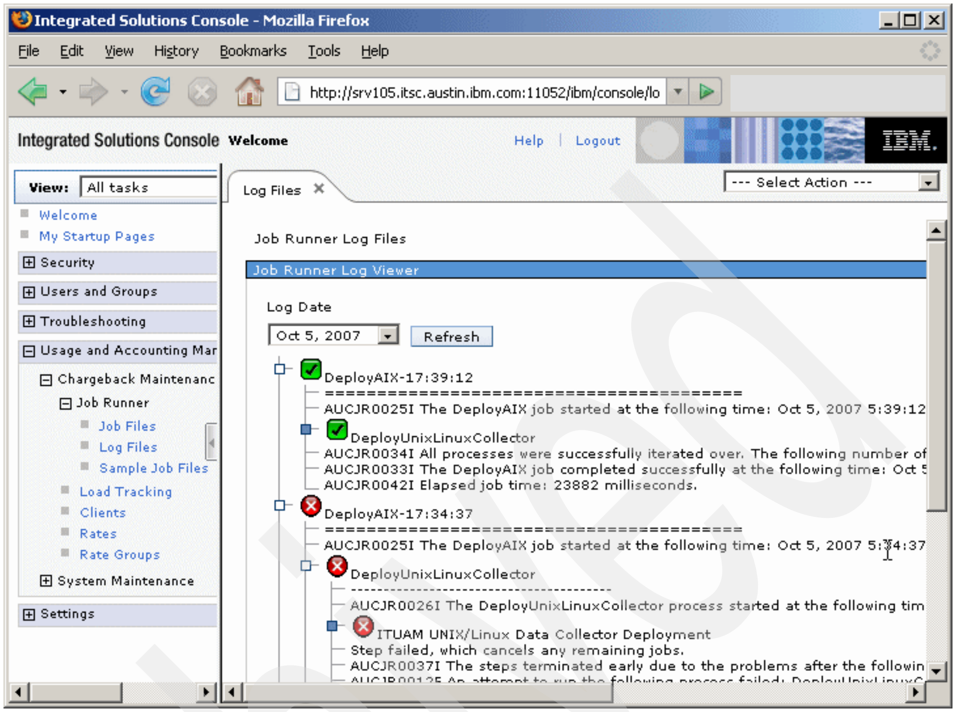


Figure 2-8 The Job Runner log viewer GUI for analyzing the detailed job output

> **Restriction:** If you set the job parameter `joblogWriteToXMLFile="false"`, you will not see a log in the Job Runner log viewer.

The job log is also available in a text format under the directory and file name of /opt/ibm/tuam/logs/jobrunner/<*JobName*>/<*date_time*>.txt. The file can be analyzed using a command as shown in Example 2-3. This can be useful when using monitoring scripts or automatic health checking.

Example 2-3 Searching the job log files on command line

```
[root@srv105 /]# cd /opt/ibm/tuam/logs/jobrunner/AIXAA_aggregated
[root@srv105 AIXAA_aggregated]# ls -tr *txt | tail -3 | while read file ; do grep -E
.*AUCJR003[1-2].* $file; done
11/5/07 13:32:11.197: INFORMATION    AUCJR0032I The job AIXAA_aggregated completed at
Nov 5, 2007 1:32:11 PM with 1 warning, 0 errors.
11/5/07 13:52:47.560: INFORMATION     AUCJR0031I The AIXAA_aggregated process completed
successfully at the following time: Nov 5, 2007 1:52:47 PM.
11/5/07 13:53:44.934: INFORMATION    AUCJR0032I The job AIXAA_aggregated completed at
Nov 5, 2007 1:53:44 PM with 0 warnings, 1 error.
```

```
[root@srv105 AIXAA_aggregated]# ls -tr *txt | tail -1 | while read file ; do echo $file
; grep -i warn $file | wc -l; grep -i error $file | wc -l ; done
20071105_135342.txt
4 # shows the # of warnings
8 # shows the # of errors
```

For detailed analysis of the last log, you can issue the command:

```
ls -tr | tail -1 | while read file; do more $file; done
```

XML structure of Job Runner

XML is a tagged file format similar to Hypertext Markup Language (HTML). XML only enforces the usage of a pair of start and end tags; the value of the tags and their attributes are enforced by the referenced extensible style document (xsd file). For Job Runner's job, the style is TUAMJobs.xsd, which is stored in the config/schemas directory of Tivoli Usage and Accounting Manager.

The structure of the XML file is shown in Figure 2-9.

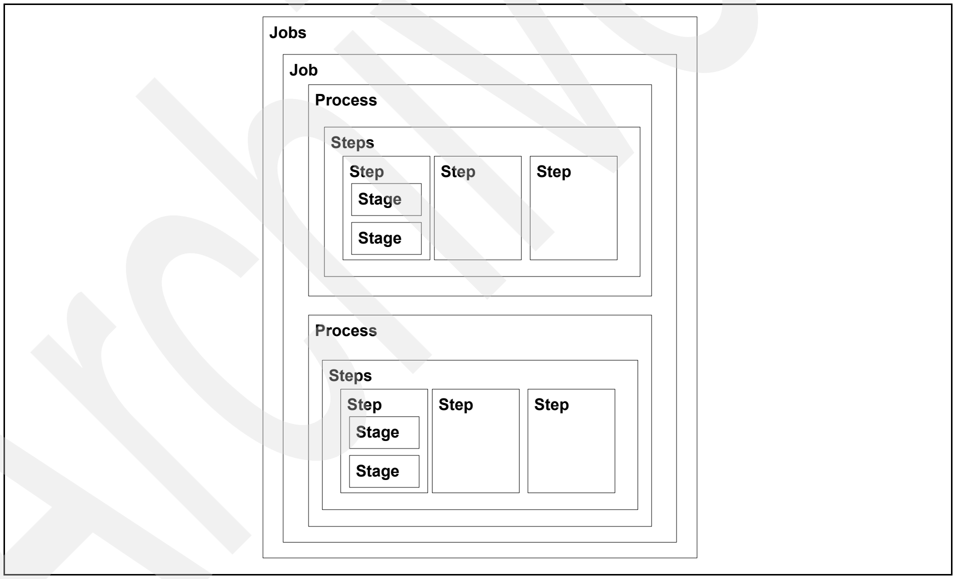


Figure 2-9 Job file structure

The components of a job file are:

Jobs This is the primary XML container for a Job Runner job file.

Job A definition of a job provides some global parameter of the job and also some e-mail notification parameters. There is typically one job per jobs.

Process A process represents sequentially processed items. You can have multiple processes within a Job. Each process group would invoke a separate Job Runner instance to run in parallel.

Steps The steps construct is a container for step items.

Step A step is the actual definition of what individual processing would be performed. A step runs an actual program. Typically a step would perform a single processing task, such as billing, scanning, sorting, cleanup, or database load; however, there is a special step called integrator that can be composed of multiple stages.

Stage A stage is a construct within the integrator step that signifies an action within the integrator step.

For detailed information about the syntax, structure, and content of each construct in the Job Runner XML, see Chapter 3, “Data collection and processing” on page 39. A typical skeleton of a Job Runner XML file is shown in Example 2-4.

Example 2-4 XML Job file skeleton

```
<Jobs...
 <Job...
  <Process id="UNIXProcessing" ....
   <Steps...
    <!-- ====================================== -->
    <!-- Step 1: Integrator with 3 stages       -->
    <!-- ====================================== -->
    <Step id="Integrator" ...
     <Integrator>
      <Input name="CollectorInput" active="true">
      </Input>
      <Stage name="CreateIdentifierFromTable"...
      </Stage>
      <Stage name="CreateIdentifierFromIdentifiers"...
      </Stage>
      <Stage name="CSRPlusOutput"...
      </Stage>
    </Step>
    <!-- ====================================== -->
    <!-- Step 2: Process using program "Bill"   -->
    <!-- ====================================== -->
    <Step id="Bill" ...
      type="Process" ...
      programName="Bill" ...
```

```
    </Step>
    <!-- ======================================= -->
    <!-- Step 3: Process using program "DBLoad"  -->
    <!-- ======================================= -->
    <Step id="DatabaseLoad" ...
      type="Process" ...
      programName="DBLoad" ...
    </Steps>
  </Process>
 </Job>
</Jobs>
```

Scheduling Job Runner job files

For regular jobs you can use the command line Job Runner statements to integrate Tivoli Usage and Accounting Manager jobs into your scheduling system shown in Example 2-5.

Example 2-5 Job Runner command line usage in scheduler definitions

```
/opt/ibm/tuam/bin/startJobRunner.sh LoadVMware.xml >> LoadVMware.log 2>&1
/opt/ibm/tuam/bin/startJobRunner.sh LoadTDSz1.xml -date today >>
LoadVMware.log 2>&1
```

The return codes for Job Runner include:

0	No warning or error
4	Warning
16	Error during processing
255	Syntax error within the parameters

Restriction: The JobRunner output in Linux goes to standard errors for all messages and standard out for exceptions only.

Or you can use the operating system scheduler, such as the crontab usage shown in Example 2-6.

Example 2-6 Job Runner crontab entries on linux

```
#
# TUAM process scheduling
#
07 6 * * * (/opt/ibm/tuam/bin/startJobRunner.sh LoadVMware.xml >>
/opt/ibm/tuam/logs/jobrunner/LoadVMware.log 2>&1)
17 6 * * * (/opt/ibm/tuam/bin/startJobRunner.sh LoadUnix.xml >>
/opt/ibm/tuam/logs/jobrunner/LoadUnix.log 2>&1)
```

```
27 6 * * * (/opt/ibm/tuam/bin/startJobRunner.sh TDSzLoad1.xml >>
/opt/ibm/tuam/logs/jobrunner/TDSzLoad1.log 2>&1)
```

2.5 Reporting accounting results

Tivoli Usage and Accounting Manager supports two reporting engines:

- 2.5.1, “Microsoft Web report viewer” on page 34
- 2.5.2, “Business Intelligence and Reporting Tools” on page 36

Note: Predefined Web reporting is available only with Microsoft Internet Information Server (IIS) and Microsoft SQL Server Reporting Services Report Viewer in Tivoli Usage and Accounting Manager Version 7.1. You can use any reporting software using SQL to generate your own reports directly from the Tivoli Usage and Accounting Manager database.

2.5.1 Microsoft Web report viewer

The reporting Web server is based on Microsoft Internet Information Server. The conceptual structure is shown in Figure 2-10.

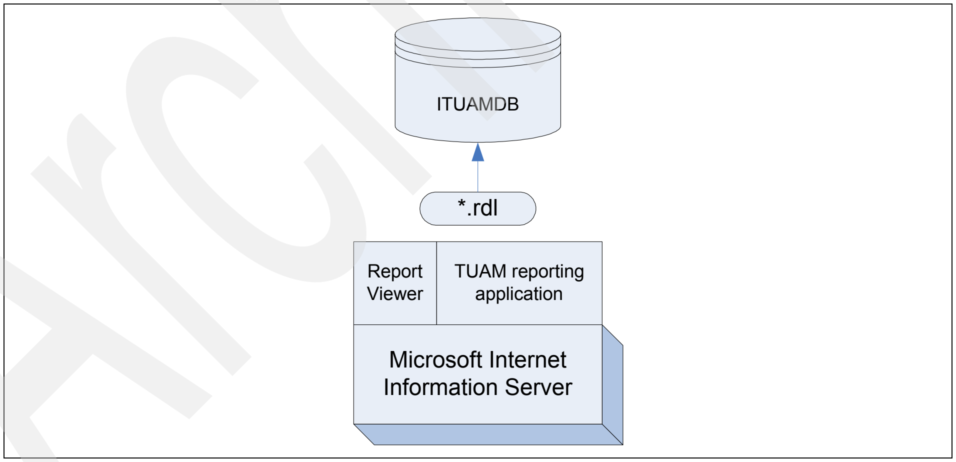


Figure 2-10 Reporting Web server structure

The reporting implementation installs an application to the Microsoft Internet Information Server based on the content in \IBM\TUAM\server\web2. The actual reports are built using Microsoft Report Server and saved as rdl files. The rdl files serving reports are installed under \IBM\TUAM\server\reportsmsrs2.

The Microsoft Internet Information Server windows of the customized application are shown in Figure 2-11.

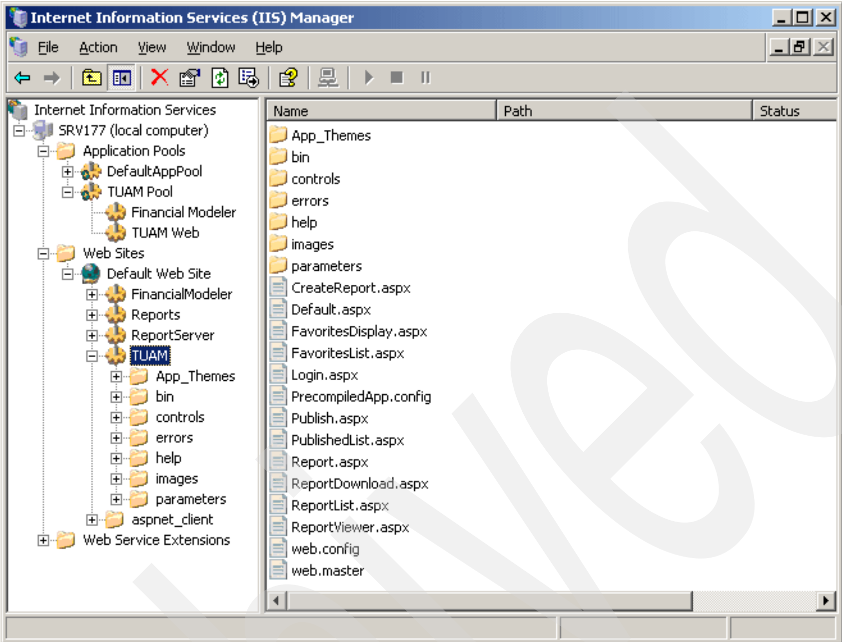
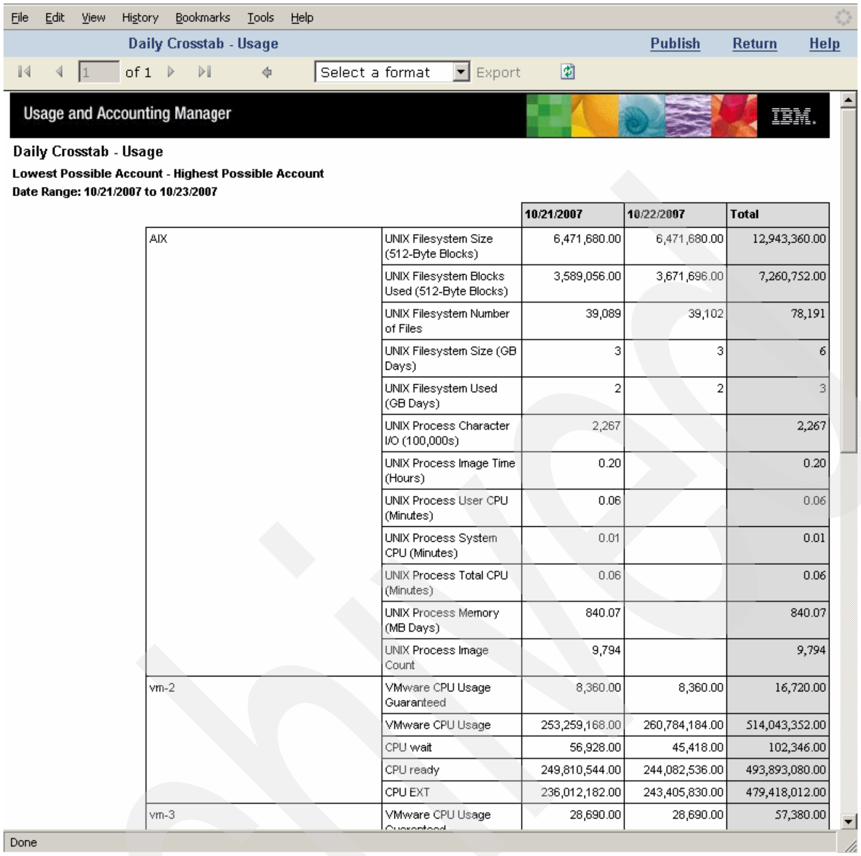


Figure 2-11 Report server setup

A sample report screen on the reporting server is shown in Figure 2-12 on page 36.



		10/21/2007	10/22/2007	Total
AIX	UNIX Filesystem Size (512-Byte Blocks)	6,471,680.00	6,471,680.00	12,943,360.00
	UNIX Filesystem Blocks Used (512-Byte Blocks)	3,589,056.00	3,671,696.00	7,260,752.00
	UNIX Filesystem Number of Files	39,089	39,102	78,191
	UNIX Filesystem Size (GB Days)	3	3	6
	UNIX Filesystem Used (GB Days)	2	2	3
	UNIX Process Character I/O (100,000s)	2,267		2,267
	UNIX Process Image Time (Hours)	0.20		0.20
	UNIX Process User CPU (Minutes)	0.06		0.06
	UNIX Process System CPU (Minutes)	0.01		0.01
	UNIX Process Total CPU (Minutes)	0.06		0.06
	UNIX Process Memory (MB Days)	840.07		840.07
	UNIX Process Image Count	9,794		9,794
vm-2	VMware CPU Usage Guaranteed	8,360.00	8,360.00	16,720.00
	VMware CPU Usage	253,259,168.00	260,784,184.00	514,043,352.00
	CPU wait	56,928.00	45,418.00	102,346.00
	CPU ready	249,810,544.00	244,082,536.00	493,893,080.00
	CPU EXT	236,012,182.00	243,405,830.00	479,418,012.00
vm-3	VMware CPU Usage Guaranteed	28,690.00	28,690.00	57,380.00

Figure 2-12 A sample report on daily usage data

2.5.2 Business Intelligence and Reporting Tools

Business Intelligence and Reporting Tools (BIRT) is an open source, Eclipse-based tool for database reporting. You can get BIRT and learn about it from the BIRT Web site:

`http://www.eclipse.org/birt/`

BIRT reporting can be viewed as shown in Figure 2-13 on page 37.

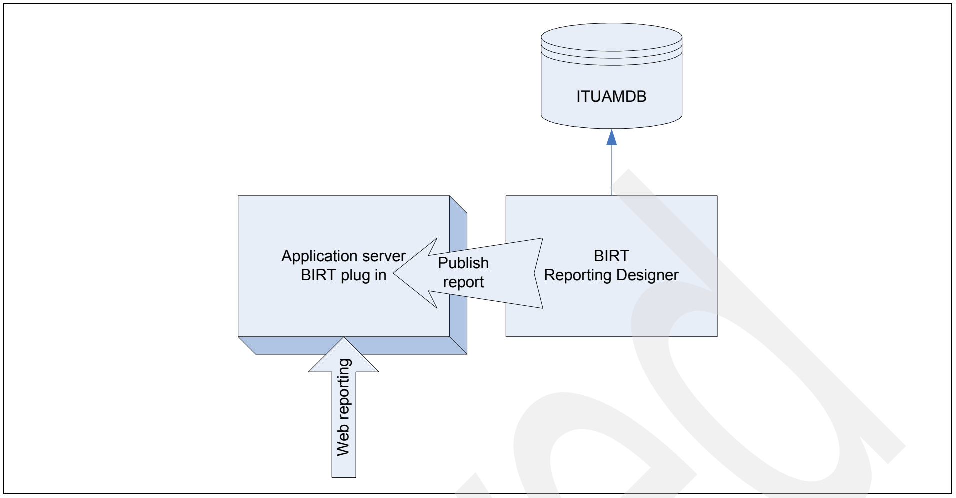


Figure 2-13 BIRT reporting

The BIRT reports must be customized using the BIRT report designer. These reports can then be run using batch commands or published through an application server that has a BIRT reporting plug in.

A sample invoice generated by BIRT is shown in Figure 2-14 on page 38.

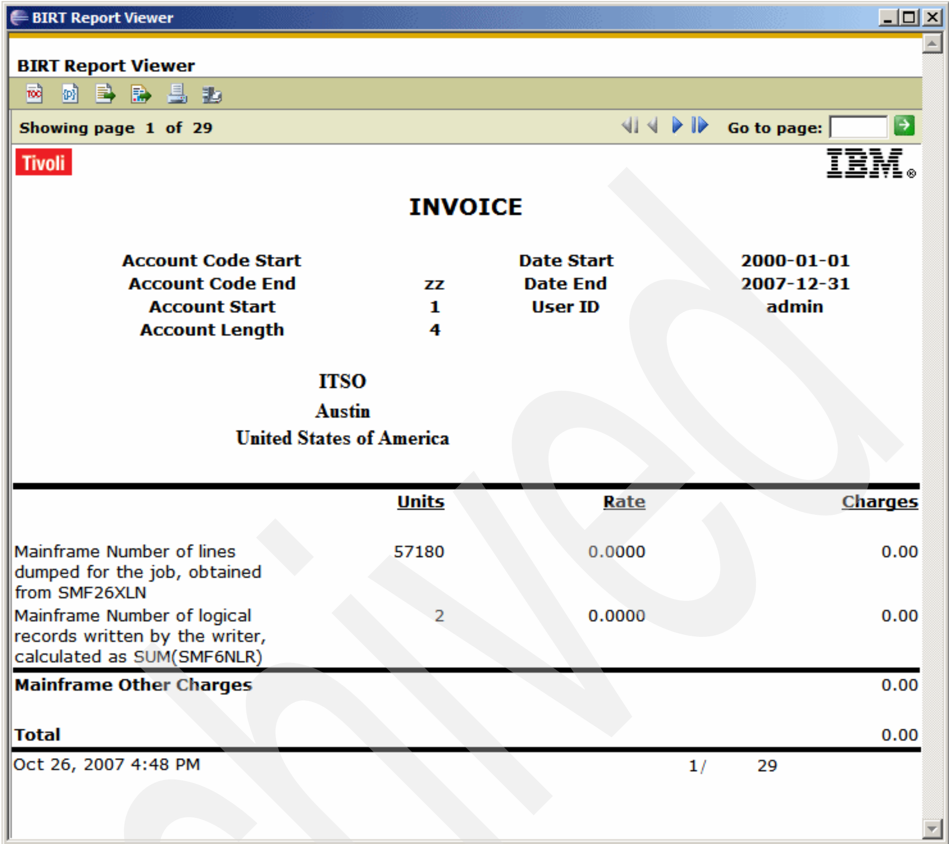


Figure 2-14 Invoice report

More on BIRT reporting setup is included in 4.9, “BIRT reporting installation and verification” on page 117.

3

Data collection and processing

This chapter discusses the Job Runner job file, including details about syntax and control. The information is based on the released version of Tivoli Usage and Accounting Manager V7.1. The following topics are covered:

3.1 Syntax of main Job Runner directives

The main Job Runner job file directives are Jobs, Job, Process, Steps, and Step. Additionally, there is the default directive that provides default parameters on each level. We explain these directives here.

3.1.1 Jobs

The `jobs` directive is the primary directive structure of the job file. It contains global directives for the whole job. Typically a job file only contains a single job, so a jobs directive is directly followed by a single job directive.

The arguments of a jobs directive are:

xmlns The name space for the XML file. You would put in `xmlns="http://www.ibm.com/TUAMJobs.xsd"` for a Tivoli Usage and Accounting Manager job. This is required for the jobs directive.

smtpServer The smtpServer to be used to send notification.

smtpFrom The indication for the source email address.

smtpTo The destination e-mail address.

smtpSubject Subject line.

smtpBody Body text.

smtpSendJobLog Boolean parameter indicates whether to include the job log in the body of the e-mail.

joblogFolder Explicitly specifies the log folder for the Job output. The default path is %HomePath%/logs/jobrunner/<jobid>.

processFolder Explicitly specifies where to write or read source and generated files. The default is under %HomePath%/processes/; this process folder is accessible by %ProcessFolder% variable.

The jobs directive can only contain the job directive.

3.1.2 Job

The `job` directive is similar to the jobs directive. The specifications on the jobs level apply; most of the time here is where you would specify these parameters.

The arguments of a job directive are:

id The name of the job (required). This name is used to determine the job log output and processing folder name under the default folder.

description A descriptive name for the job.

dataSourceId Optional data source ID that specifies which database to use for the job database connection, including for the configuration settings and loading data. The default value is the Default.Processing data source as specified in the local registry.tuam.xml file.

processPriorityClass Priority of the job.

stopOnProcessFailure Whether or not to stop the processing if a step failed.

active Whether or not the job is active.

joblogShowStepOutput Whether or not to write the step output to the job log result.

joblogShowStepParameters Whether or not to show the step parameter in the job log output.

joblogWriteToDB This is not implemented in Tivoli Usage and Accounting Manager V7.1.

joblogWriteToTextFile Whether or not a text file is created in the job log output directory.

joblogWriteToXMLFile Whether or not an XML file is created in the job log output directory. The ISC uses only the XML file for displaying the output; if this option is set to false, you cannot see the result from ISC.

smtpServer The smtpServer to be used to send notification.

smtpFrom The indication for the source e-mail address.

smtpTo The destination e-mail address.

smtpSubject Subject line.

smtpBody Body text.

smtpSendJobLog Boolean parameter indicating whether to include the job log in the body of the e-mail.

joblogFolder Explicitly specifies the log folder for the Job output. The default path is %HomePath%/logs/jobrunner/<jobid>.

processFolder Explicitly specifies where to write or read source and generated files. The default is under %HomePath%/samples/processes/; this process folder is accessible by %ProcessFolder% variable.

The job directive can contain the following directives:

- Defaults: Provides additional default parameters for the job elements.
- Process: This is the most common direct node under the job directive. You can have multiple processes that execute in parallel.
- Steps: Collection of step directives that must be executed sequentially for this job. This is typically put under a process directive.

3.1.3 Defaults

The `defaults` directive provides name-value pairs for the containing level (job or process or step). The defaults directive can contain any name and value pairs as arguments.

The defaults directive can contain the default directive. The default directive has the attribute name and value that can also be used to set any default name and value pairs. The reserved names that have special processing functions are: LogDate, RetentionFlag, and programName.

3.1.4 Process

The `process` directive is used to signify a collection of steps directives that must be executed sequentially. It is common practice to put a process directive under job (instead of the steps directly) to allow flexibility on adding other process directives.

The arguments of a process directive are:

id The name of the process (required).
description A descriptive name for the process.
processPriorityClass Process priority.
buildProcessFolder Whether or not the process folder is created if it does not already exist.
joblogShowStepOutput Whether or not to write the step output to the job log result.
joblogShowStepParameters Whether or not to show the step parameter in the job log output.
active Whether or not to execute this process.

The process directive can contain the following directives:

- Defaults
- Steps

3.1.5 Steps

The `steps` directive is a container for step directives. It only has a single argument, which is:

stopOnStepFailure Whether or not to stop the execution if a step is considered failed or in error.

The steps directive can only contain the step directive.

3.1.6 Step

The `step` directive is the main specification of what the job is doing. It can be written differently depending on the function you are invoking. Step writing and processing are discussed in the next section. A step also can be invoked directly using the StepRunner java program with the argument of nodename, programName, and programType.

The arguments for the step directive are:

id A unique name for the step is required.
description A descriptive name for the step.
programName The name of the program.
type Program type. The only acceptable values are ConvertToCSR or process.
programType The type of program. Some types are java, wsf, and console.
processPriorityClass Step priority.
buildProcessFolder This provides an override for the build process folder.
joblogShowStepOutput Whether or not to write the step output to the job log result.
joblogShowStepParameters Whether or not to show the step parameter in the job log output.
active Whether or not to execute this step.

The combination of *only* programName and programType determines the program or action that this step invokes.

The step can contain the following directives:

- Parameters: Collection of step parameters.
- Step specific directives depending on the programName and programType combination. Each of these directives is mutually exclusive; they are only used corresponding to the program invoked:
  - integrator
  - generateexternalfile
  - acct
  - bill
  - dbload
  - dbpurge
  - jobfileconversion
  - generatexmlfile
  - cimswindisk or windisk
  - cimswineventlog or wineventlog

  The integrator program is discussed in 3.3, “The integrator program” on page 52; the other step specific directives are covered in the next section.

3.2 Writing a step

The step is mainly governed by the progamType and programName attributes. These attributes determine the way processing is performed. Our tests indicate that the type attribute, though it is required, does not affect processing. Table 3-1 shows the combinations of programName and programType and the associated program each combination actually invokes. Other combinations may exist for maintaining backward compatibilities.

Table 3-1 Program combinations

programType	programName	Invoked program
java	Integrator	integrator.StepRunIntegrator
java	SendMail	mail.StepRunMail
java	Acct	acct.StepRunAcct
java	Bill	bill.StepRunBill
java	Sort	sort.StepRunSort
java	DBLoad	load.StepRunLoad
java	DBPurge	purge.StepRunPurge
java	JobFileConversion	jobfileconversion.StepRunJBConversion
java	Rpd	rpd.StepRunRpd
java	Scan	scan.StepRunScan
java	Cleanup	cleanup.StepRunCleanupScan
java	FileTransfer	filetransfer.StepRunFileTransfer
java	WaitFile	waitfile.StepRunWaitFile
java	<java program name>	java.StepRunJava
Console	<program name>	console.StepRunConsole
WSF	<wsf script name>	wsf.StepRunWSF
java	SingleProcessStep[a]	StepRunAcct - StepRunSort - StepRunBill

a. A special step that has the programName of singleprocessstep and programType of java would generate an automatic job with a set of accounting process (StepRunAcct), sorting (StepRunSort), and billing (StepRunBill) steps.

All the directives under steps have the following general attributes:

joblogShowStepOutput Whether or not to write the step output to the job log result.

joblogShowStepParameters Whether or not to show the step parameter in the job log output.

processFolder Provides an override for the process folder. This does not apply to the mail program. Note that scan, sort CleanUp, Scan, FileTransfer, and WaitFile use ProcessFolder (an *uppercase* P).

dataSourceId Provides an override for data source access in the associated step. This is only used for Load, Purge and JobConversion.

dbConfigurationFile Database configuration file for Acct and Bill. This does not seem to be used.

In the following sections we describe the detailed parameters and processing function of each invoked program.

3.2.1 Mail

The mail step allows you to send an e-mail message. The applicable parameters are:

SMTPServer The smtpServer to be used to send notification

FromEmail The indication for the source e-mail address

ToEmail The destination e-mail address

Subject Subject line

Body Content of the e-mail

AttachFileLocation Files that you attach to the e-mail

3.2.2 Acct

The acct process derives account code information based on an account code lookup file (Accttbl.txt). This program is provided as a backward compatibility option for existing jobs. The newer recommended method is to use the integrator step's CreateIdentifierFromTable stage to derive a new identifier for the account code field.

Before the acct program is run, a temporary step XML file is created. The file is created in the ProcessFolder with the step's ID as a name. This program can be invoked directly using the Java class com.ibm.tuam.acct.AcctMain and supplying <processFolder> and step.xml as arguments.

The applicable parameters are:

inputFile Input filename (default: CurrentCSR.txt)
outputFile Output filename (default: AcctCSR.txt)
inputFileEncoding Encoding of input file
outputFileEncoding Encoding of output file
controlFile Control parameters (default: AcctCntl.txt)
exceptionFile Exception file name (records that cannot be matched)
ControlCard Content of control file in line
ProcessFolder Processing folder name
Trace Set tracing to true or false
accCodeConvTable Account code conversion table text file (default: Accttbl.txt)
accCodeConvTableEncoding Encoding of the conversion table file

The account process is illustrated in Figure 3-1.

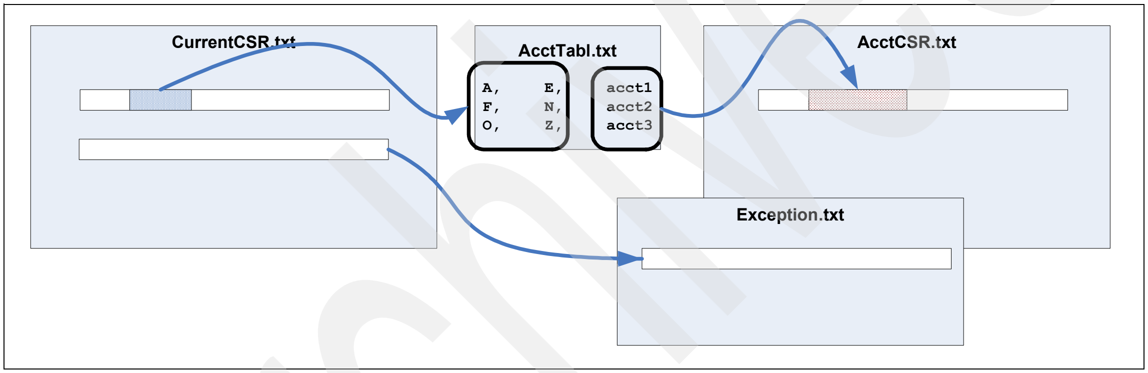


Figure 3-1 Account process

3.2.3 Bill

The bill process processes the usage data that already have account codes to generate billing information. As discussed in 2.4.3, “Account code and rate” on page 24, account code is used to get the rate table and find the rate for each resource in the CSR file. The result of this process is three files:

BillSummary.txt Billing summary in which each account and resource has the usage and money value shown

BillDetail.txt Billing detail in which records are provided containing account code and usage entries with reference to the identifier.

Ident.txt Identifiers referred to from the billing files.

Before the bill program is run, a temporary step XML file is created. The file is created in the ProcessFolder with the step's ID as its name. This program can be invoked directly using the Java class com.ibm.tuam.bill.BillMain and supplying <processFolder> and step.xml as arguments.

The applicable parameter is:

inputFile Input filename (default: AcctCSR.txt). CSR+ files are recommended because they can be sorted by the billing program for optimization.

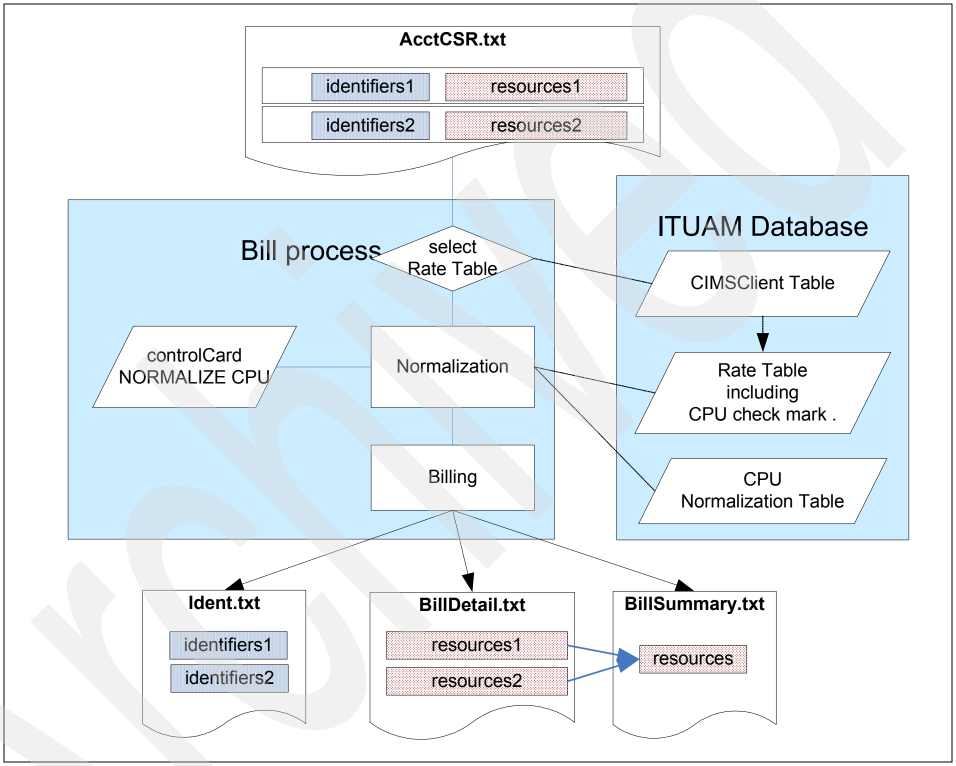


Figure 3-2 The bill process overview

> **Note:** It is recommended that you use a CSR file sorted by the account code as the input file, or a CSR+ file, because this is sorted by the billing automatically.

3.2.4 Cleanup

The cleanup step defines the option to clean up the processing directory for older files. It can remove the files either by date or by age. The applicable parameters are:

DaysToRetainFiles The age of files to retain, DaysToRetainFiles and DateToRetainFiles are mutually exclusive. If both are specified, DaysToRetainFiles will be used.

DateToRetainFiles The date that denotes the oldest file to retain.

Folder Folder name to clean up, default is the ProcessFolder/jobid.

CleanSubFolders Whether or not to clean all sub folders.

3.2.5 Sort

The sort step provides a mechanism for sorting a CSR file line by line. This sort program does not allows you to select the identifiers to sort on. The applicable parameters are:

InputFileName Input file, default is AcctCSR.txt

OutputFileName Output file name, default is AcctCSR.txt

> **Tip:** For a more specific sort, based on identifiers, use the Integrator sort described in 3.3, “The integrator program” on page 52.

3.2.6 DBLoad

The load step provides the capability to load data into the database. Data is typically loaded from billing output files.

Before the database load program is run, a temporary step XML file is created. The file is created in the processFolder with the step’s ID as its name. This program can be invoked directly using the Java class com.ibm.tuam.load.DBLoadMain and supplying <processFolder> and step.xml as arguments.

The applicable parameters are:

resourceFile File name for the resource data

processFolder Folder to get the files

Trace Whether or not to perform tracing

detailFile File name for the detail data

allowDetailDuplicates Whether or not to allow duplicate loads on the detail file

summaryFile File name for the summary data

allowSummaryDuplicates Whether or not to allow duplicate loads on the summary file
onEmptyFile Status to indicate if an empty file is found; possible values are Success, Warning, or Fail
identFile File name for the identifier
encoding File encoding
loadType Detail, Summary, Ident, All, Resource
bulkLoad Whether to invoke the load as a bulk process

3.2.7 DBPurge

The purge step provides the method to purge data in the Tivoli Usage and Accounting Manager database.

Before the purge program is run, a temporary step XML file is created. The file is created in the processFolder with the step's ID as its name. This program can be invoked directly using the Java class com.ibm.tuam.purge.DBPurgeMain and supplying <processFolder> and step.xml as arguments.

The applicable parameters are:

MonthsToKeep Number of months to retain data; if specified, the start and end date are ignored
StartDate Optional start date for purging
EndDate Optional end date for purging
PurgeSummary Whether or not to purge the summary table
PurgeBillDetail Whether or not to purge the billing detail table
PurgeIdent Whether or not to purge the identifier table
PurgeAcctDetail Whether or not to purge the account detail table
PurgeClient Whether or not to purge the client table
PurgeRate Whether or not to purge the rate table
DataSourceID ID of the data source to be accessed

3.2.8 Scan

The scan step provides the facility of merging multiple CSR files in the processing folder into a single CSR file. The applicable parameters are:

UseLogDateRange Log date range to be merged.
LogDate A single log date to merge.
RetainFileDate Determine the output file name. If RetainFileDate is true, the file name is set to the end date of the log date. If false, the output filename is CurrentCSR.txt.
RetainDateFlag Whether or not to retain the file date (same as RetainFileDate).

ExcludeFile	Exclude specified files from merge.
ExcludeFolder	Exclude specified folders from merge.
IncludeFile	Include specific files.
UseStepFiles	Use files generated from previous steps.
AllowMissingFiles	Allow processing to proceed even when there are files that are missing.
AllowEmptyFiles	Allow files to be empty.

3.2.9 File transfer

The file transfer program provides the facility to perform file transfer between systems. It provides several different methods for performing file transfer, such as ftp, scp, sftp or smb.

The applicable parameters are:

continueOnError	Whether to stop or continue when error occurs
type	Transfer type; keywords are: ftp, file, win, windows, ssh, rsh, rexec
overwrite	Whether to overwrite the file if it is already there
ServerName	The target server name
UserId	The user ID to be used
UserPassword	The password to be used
from*	Source directories
to*	Target directories

Some additional parameters for ftp:

OpenType®	FTP site type
TransferType	ASCII or binary transfer

Additional parameter for secure transfer:

KeyStoreFileName	File to store SSL certificates

3.2.10 Wait file

This waitfile step waits until a file becomes available. The applicable parameters are:

FileName	File name to be waited for.
PollingInterval	The duration between checking for the file.
TimeOut	The total time to wait for the file; this is the value that is used if both TimeOut and TimeOutDateTime exists.
TimeOutDateTime	The time stamp when the program should stop waiting.

3.2.11 Remote product deployment

The remote product deployment (rpd) provides a mechanism to perform file transfer and product installation. This program is mainly used for deployment of collector and AIX advanced accounting collection. This uses the same mechanism as the File transfer in 3.2.9, "File transfer" on page 50.

The applicable arguments are:

Host	Target host.
UserId	User ID to use.
Password	Password to use.
Manifest	The XML file name that describes the action sequences to perform.
RPDParameters	Various parameters for RPD program as requested by the Manifest. This is in the form of keyword value pairs.
SourcePath	The path to where the files to be transferred reside.

Note: We use rpd for transferring AIX Virtual I/O Server data and deploying UNIX collector. See 6.1.2, "Virtual I/O server data collection" on page 161 and 5.2.1, "Remote installation of the AIX data collector" on page 129.

3.2.12 Job conversion

This job conversion step is a stand alone Job runner step for converting Tivoli Usage and Accounting Manager V6.x job files into a V7.1 job format. The applicable parameters are:

inputFolder	Input conversion folder
outputFolder	Output conversion folder
overWriteOutputFolder	Whether to overwrite any files that already exist

Note: There is another conversion for old Conversion Builder files into an Integrator stage, which is not covered here. The Integrator conversion of input files is discussed in 3.4.1, "The Job Runner integrator collector" on page 56.

3.2.13 Windows script file

The Windows script file (wsf) step is a special step that would run Windows Script File. It typically contains a Microsoft Visual Basic® program. The execution is using `csript.exe` command. The Visual Basic program is embedded in the programName argument of the step. The parameters for this step are passed to the script directly.

3.2.14 Java

The Java step allows you to run an arbitrary Java program. This is typically to invoke your own Java function or conversion step. You must supply the appropriate library and command line options for the Java program. The applicable parameters are:

UseCommandProcessor Whether to invoke the command processor or invoke the Java class directly
UseStandardParameters Whether to use standard JVM™ parameters
JavaCommandLine Command line argument for the Java Virtual Machine
JavaHome Home directory of the Java executable

3.2.15 Console

The console step allows you to invoke a program on the operating system level. The applicable parameters is:

UseCommandProcessor Whether to invoke the command processor

3.3 The integrator program

The integrator program is a specialized Java program that can contain multiple stages. This program is used to manipulate and convert a CSR or CSR+ formatted file. The first stage is an input definition; the last stage is called CSROutput. Each stage of the integrator program is a record processor, which processes each record and passes it to the next stage, similar to pipeline processing.

An Integrator step has some required and some optional sections:

- <Input> is required, along with at least one of the following elements:
  – <Collector>
  – <Parameter>
  – <File>
- <Stage name="*function*"> processing stage; include as many as needed
- <Stage name="CSROutput"> or <Stage name="CSRPlusOutput"> is required

The structure of an integrator step is illustrated in Figure 3-3 on page 53.

```
<Step id=”integrator” programType=”java” type=”Process”>
   <Integrator>
      <Input name="CSRInput">
         <Files><file name=”CSRinput.txt” /></files>
      </Input>
      <Stage name="CreateIdentifierFromValue" active="true">
      <Stage name="CreateIdentifierFromRegEx" active="true">
      <Stage name="Aggregator" active="true"></Stage>
         <Parameters><Parameter defaultAggregation="false" />
         </Parameters>
      </Stage>
      <Stage name="ResourceConversion" active="true">
      <Stage name="CreateResourceFromValue" active="true">
      <Stage name="RenameFields" active="true">
      <Stage name="CreateIdentifierFromTable" active="true"></Stage>
      <Stage name="CreateIdentifierFromIdentifiers" active="true">
      <Stage name="DropFields" active="true">
      <Stage name="Sort" active="true"></Stage>
   <!-- the next stage is required as last stage of integrator -->
      <Stage name="CSROutput" active="true">
         <Files><File name="CSRoutput.txt"/></Files>
      </Stage>
   </Integrator>
</Step>
```

Figure 3-3 A selection of the most common ConvertToCSR integrator functions

The processing of the integrator step depicted in Figure 3-3 is shown in Figure 3-4.

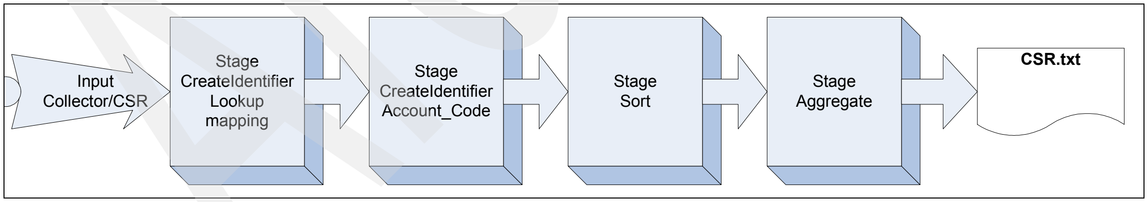


Figure 3-4 Integrator processing

The following sections discuss the stages of the integrator program.

3.3.1 Input

You can choose one out of several types for input.

The keywords for input names are:

AIXAAInput	AIX Advanced Accounting
ApacheCommonLogFormat	Apache HTTP server common log
CollectorInput	Specifying a collector program by the Collector name
CSRInput	Standard CSR file
NCSAInput	WebSphere collector
NotesDatabaseSizeInput	Lotus Notes collector
NotesEmailInput	Lotus Notes collector
NotesUsageInput	Lotus Notes collector
W3CWinLog	Microsoft Internet Information Server collector

Specifically for the CollectorInput type, you can have the collector defined as follows:

DATABASE	Database
DELIMITED	Comma or tab delimited file
EXCHANGE2007	Microsoft Exchange server
FIXEDFIELD	Fixed field files
TDS	Tivoli Decision Support for z/OS database
TPC	TotalStorage Productivity Center
TRANSACTION	CIMSTransaction table converter
VMWARE	Web Services SDK call to VMware Virtual Center
WEBSPHEREXDFINEGRAIN	WebSphere collector
WEBSPHEREXDSERVER	WebSphere collector

To specify the default folder for transferred collector files we use the variable %CollectorLogs% in the path definition.

3.3.2 Processing

The processing stages can be read from the <TUAM reference>. The stage functions are:

- Aggregator
  Merge data on which all the identifiers are the same and merge into a single record by summarizing the resource values.
- CreateIdentifierFromIdentifiers
  Copy, parse, and merge identifiers into a new one.
- CreateIdentifierFromRegEx
  Copy, parse, and merge identifiers using a regular expression.

- CreateIdentifierFromTable
  Use a text lookup table to search for a matching entry and put it into an identifier.
- CreateIdentifierFromValue
  Write an identifier from a fixed string.
- CreateResourceFromConversion
  Calculate a new resource from other resources.
- CreateResourceFromValue
  Create a fixed value resource. (We can create resource names longer than eight characters, but once we want to define them as a rate, the limitation to eight characters will prevent us from using them.)
- DropFields
  Drop a field, either an identifier or a resource.
- DropIdentifiers
  Drop identifiers field.
- DropResources
  Drop resources field.
- ExcludeRecsByDate
  Filter some records by a certain date.
- ExcludeRecsByPresence
  Filter some records by presence of a field.
- ExcludeRecsByValue
  Filter some records by value for a field.
- IdentifierConversionFromTable
  Change an identifier using the conversion from the table.
- IncludeRecsByDate
  Filter some records that are not on a certain date.
- IncludeRecsByPresence
  Filter some records that do not have a certain field.
- IncludeRecsByValue
  Only get the records that have a certain value.
- MaxRecords
  Include only a specific number of records from the CSR file, best for debugging a collection job.
- RenameFields
  Rename an identifier or a resource in bulk.
- ResourceConversion
  Calculate a new value for a resource based on one or more resource values.

- Sort
  Sort the CSR file records based on certain identifier values.

3.3.3 Output

The output types can only be CSROutput or CSRPlusOutput format. It encloses a Files directive with a single File directive. A sample Output stage is shown in Example 3-1.

Example 3-1 Sample CSRPlusOutput stage

```
<Stage name="CSRPlusOutput" active="true">
   <Files>
      <File name="%ProcessFolder%/server1/%LogDate_End%.txt" />
   </Files>
</Stage>
```

3.4 Using integrator jobs

This section describes some practical integrator job uses:

- 3.4.1, “The Job Runner integrator collector” on page 56
- 3.4.2, “Account conversion using the Job Runner Integrator” on page 60

3.4.1 The Job Runner integrator collector

The Input for the integrator jobs can be read directly from any text file or database using the collector function. In this section, we discuss two of the most common ways to extend Tivoli Usage and Accounting Manager processing by collecting from a delimited file or a database query.

In either case, the source data has to resemble a table, meaning with rows of individual data and columns of fields. The fields are then mapped into either header, identifier, or resource fields of a CSR record.

The collector input processing is illustrated in Figure 3-5 on page 57.

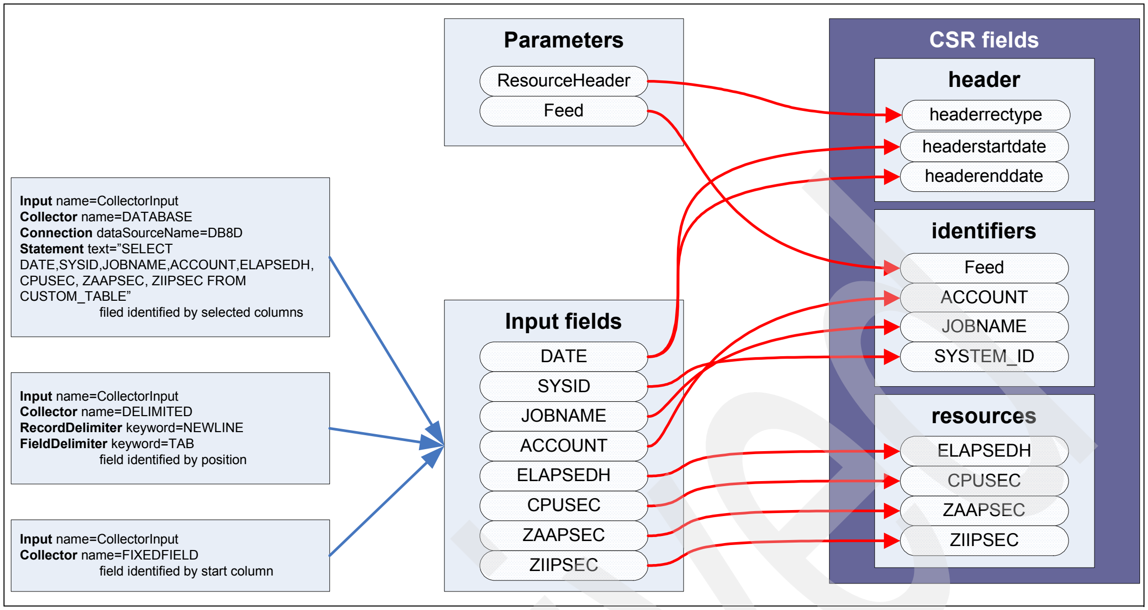


Figure 3-5 Collector input processing

A sample collector definition that implemented the structure in Figure 3-5 using database query is shown in Example 3-2.

Example 3-2 Sample collector input for database

```
<Input name="CollectorInput" active="true">
   <Collector name="DATABASE">
   <Connection dataSourceName="DB8D" />
   <Statement text="SELECT DATE, SYSID, JOBNAME,
                    ACCOUNT, ELAPSEDH, CPUSEC, ZAAPSEC, ZIIPSEC
                    FROM CUSTOM_TABLE" />
      <Parameter src="PARAMETER" sqlType="DATE"
         position="1" srcName="StartLogDate" />
      <Parameter src="PARAMETER" sqlType="DATE"
         position="2" srcName="EndLogDate" />
   </Collector>
   <Parameters>
      <Parameter name="StartLogDate" value="%LogDate_Start%"
         dataType="DATETIME" format="yyyyMMdd" />
      <Parameter name="EndLogDate" value="%LogDate_End%"
         dataType="DATETIME" format="yyyyMMdd" />
      <Parameter name="Resourceheader" Value="TDSzUSER"
         dataType="STRING" />
```

```
      <Parameter name="Feed" value="SC67" dataType="STRING" />
      <Parameter name="LogDate" value="%LogDate_End%"
         dataType="DATETIME" format="yyyyMMdd" />
   </Parameters>
```

The collector definition must then be defined as input fields. For SQL, this should match the number of columns defined in the select statement.

The input field definitions are shown in Example 3-3.

Example 3-3 Input fields

```
   <InputFields>
      <InputField name=”1” columnName=”DATE”
         dataType=”DATETIME” format=”yyyyMMdd”/>
      <InputField name=”2” columnName=”SYSID” dataType=”STING”/>
      <InputField name=”3” columnName=”JOBNAME” dataType=”DOUBLE”/>
      <InputField name=”4” columnName=”ACCOUNT” dataType=”DOUBLE”/>
      <InputField name=”5” columnName=”TAPEBLKS” dataType=”DOUBLE”/>
      <InputField name=”6” columnName=”ELAPSEPDH” dataType=”DOUBLE”/>
      <InputField name=”7” columnName=”CPUSEC” dataType=”STRING”/>
      <InputField name=”8” columnName=”ZAAPSEC” dataType=”STRING”/>
      <InputField name=”9” columnName=”ZIIPSEC” dataType=”STRING”/>
   </InputFields>
```

The output field definitions are shown in Example 3-4. If the source is INPUT, the srcName refers to the name field of the InputField definition in Example 3-3.

Example 3-4 Output fields

```
<OutputFields>
   <OutputField name=”headerrectype” src=”PARAMETER”
      srcName=”ResourceHeader” />
   <OutputField name=”headerstartdate” src=”INPUT” srcName=”1” />
   <OutputField name=”headerenddate” src=”INPUT” srcName=”1” />
   <OutputField name=”Feed” src=”PARAMETER” srcName=”Feed” />
   <OutputField name=”ACCOUNT” src=”INPUT” srcName=”4” />
   <OutputField name=”JOBNAME” src=”INPUT” srcName=”3” />
   <OutputField name=”SYSTEMID” src=”INPUT” srcName=”2” />
   <OutputField name=”ELAPSEDH” src=”INPUT” srcName=”6” />
   <OutputField name=”CPUSEC” src=”INPUT” srcName=”7” />
   <OutputField name=”ZAAPSEC” src=”INPUT” srcName=”8” />
   <OutputField name=”ZIPSEC” src=”INPUT” srcName=”9” />
</OutputFields>
```

Conversion from a delimited or fixed format text used to be supported using the Conversion Builder. This is no longer shipped with Tivoli Usage and Accounting Manager V7.1. We can use integrator as illustrated in Example 3-5.

Example 3-5 Integrator conversion

```
<Integrator>
   <Input active="true" name="CollectorInput">
      <Collector name="DELIMITED">
         <RecordDelimiter keyword="NEWLINE"/>
         <FieldDelimiter keyword="COMMA"/>
         <TextFieldQualifier keyword="NONE"/>
      </Collector>
      <Parameters>
         <Parameter name="UnivHdr" value="ITUAMDBsize"/>
         <Parameter name="DATE" value="%LogDate_End%"
            format="yyyyMMdd"/>
      </Parameters>
      <InputFields>
         <InputField dataType="STRING" name="TABLE_NAME" position="1"/>
         <InputField dataType="STRING" name="ROWS" position="2"/>
         <InputField dataType="STRING" name="SIZE_KB" position="3"/>
      </InputFields>
      <OutputFields>
         <OutputField name="headerrectype" src="PARAMETER"
            srcName="UnivHdr"/>
         <OutputField dateKeyword="SYSDATE" name="headerstartdate"
            src="KEYWORD" timeKeyword="SYSTIME"/>
         <OutputField dateKeyword="SYSDATE" name="headerenddate"
            src="KEYWORD" timeKeyword="SYSTIME"/>
         <OutputField name="TABLE_NAME" src="INPUT"
            srcName="TABLE_NAME"/>
         <OutputField name="ROWS" resource="true" src="INPUT"
            srcName="ROWS"/>
         <OutputField name="SIZE_KB" resource="true" src="INPUT"
            srcName="SIZE_KB"/>
      </OutputFields>
      <Files>
         <File name="%processFolder%/%LogDate_End%-spreport.csv"
            type="input"/>
      </Files>
   </Input>
   .  .  .
<!-- put further stage(s) and required CSROutput stage in here -->
   .  .  .
</Integrator>
```

Restriction: The Tivoli Usage and Accounting Manager Conversion Builder is no longer shipped and not all functions can be directly transferred into an integrator process automatically using the Definition File Conversion tool. One or more stages and additional steps might be needed to implement the Filter and Parse function of the conversion builder.

You should now understand how to make a user-defined collector from arbitrary data.

3.4.2 Account conversion using the Job Runner Integrator

Matching CSR record identifiers from the collected source into proper account codes that adhere to the account code structure is necessary for proper reporting. This can be done once we have the data in the CSR format, typically from the input collection discussed in the previous section.

The recommended way of creating the Account_Code field is using the Integrator. The older program, called Acct, is still valid for backward compatibility, but as you roll out new components it should become obsolete in favor of the integrator account conversion.

Figure 3-6 on page 61 shows the overview of Account Code conversion.

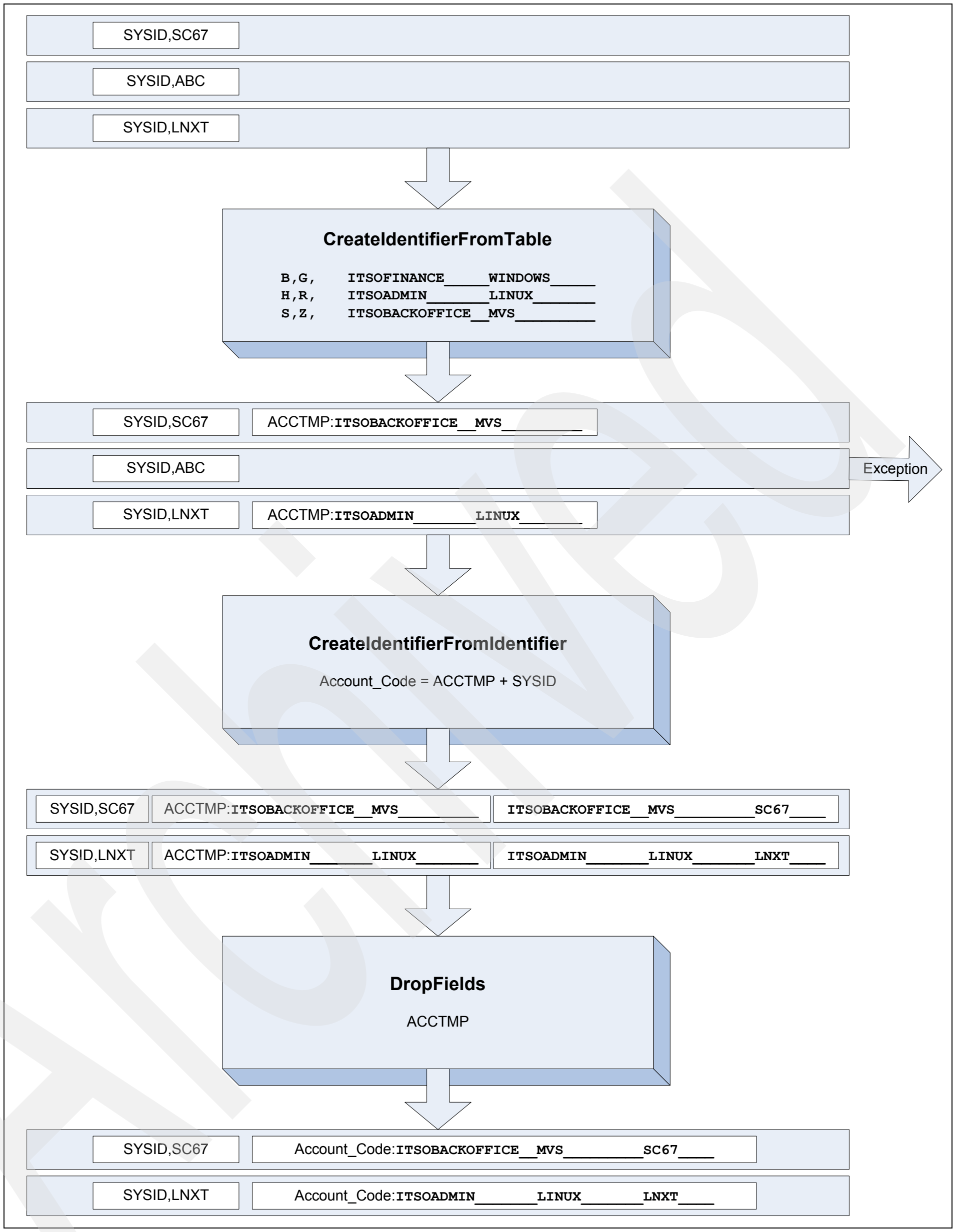


Figure 3-6 The Account Code conversion using Integrator

In Figure 3-6, we show that the SYSID is used as the key to look up the value of the ACCTMP field based on the conversion table (AcctTabl.txt). If there is a record that does not have a matching result, the record is sent to the exception file. The next stage appends the SYSID to ACCTMP to create the Account_Code identifier. Then we can remove the ACCTMP field. Example 3-6 shows the integrator implementation illustrated in Figure 3-6.

Example 3-6 Account Conversion using Job Runner Integrator

```
<!--  get account code from table based on SYSID (hostname) -->
<Stage name="CreateIdentifierFromTable" active="true">
   <Identifiers>
      <Identifier name="ACCTMP">
         <FromIdentifiers>
            <FromIdentifier name="SYSID" offset="1" length="10"/>
         </FromIdentifiers>
      </Identifier>
   </Identifiers>
   <Files>
      <File name="/opt/ibm/tuam/processes/Accttabl.txt" type="table"/>
      <File name="Exception-%LogDate_End%.txt" type="exception"
         format="CSROutput"/>
   </Files>
   <Parameters>
      <!-- exception and writeNoMatch should be set as such -->
      <Parameter exceptionProcess="true"/>
      <Parameter writeNoMatch="false"/>
      <Parameter sort="true"/>
      <Parameter upperCase="false"/>
      <Parameter modifyIfExists="true"/>
   </Parameters>
</Stage>
<!--  add hostname as last part to the account code -->
<Stage name="CreateIdentifierFromIdentifiers" active="true">
   <Identifiers>
      <Identifier name="Account_Code">
         <FromIdentifiers>
           <FromIdentifier name="ACCTMP" offset="1" length="28"/>
           <FromIdentifier name="SYSID" offset="1" length="8"/>
         </FromIdentifiers>
      </Identifier>
   </Identifiers>
   <Parameters>
      <Parameter modifyIfExists="true"/>
      <Parameter keepLength="true"/>
   </Parameters>
</Stage>
<!--  drop temporary account code identifier -->
```

```
<Stage name="DropFields" active="true">
   <Fields>
      <Field name="ACCTMP"/>
   </Fields>
</Stage>
```

Using the CreateIdentifierFromTable function, all unmatched identifiers will be collected in a exception file, keeping the structure of the input file for this step.

The file specified will be overwritten unless you add a variable %LogDate_End% to the definition. There will be no accumulation from the last processing, and we recommend a separate reprocessing job. Otherwise you would need to unload all the data and reprocess the complete job once the account table is updated.

3.5 Scheduling jobs using Tivoli Workload Scheduler

In most cases, we scheduled our jobs using the basic mechanism from the operating systems, such as AT service in Windows or crontab in UNIX systems (see "Scheduling Job Runner job files" on page 33).

While this mechanism worked well in our small scale test environment, a real production implementation requires a bit more features, such as the ability to:

- Schedule and synchronize the schedules across several machines
- Provide a return code processing, and automatic recovery from failures
- Run several jobs in series, possibly on different machines

As demonstrated in this section, these functions are available using a job scheduler subsystem, which is an extension to IBM Tivoli Workload Scheduler. While you can run the startJobRunner directly from the command line, we would like to demonstrate an extended agent implementation for running Tivoli Usage and Accounting Manager job runner's jobs.

Tivoli Workload Scheduler is a job scheduling solution that allows job schedules to be defined and run within your enterprise. It is controlled from the Master Domain Manager, and the agents are called the Fault Tolerant Agents. It is called Fault Tolerant Agents because job schedules continue to run, even when the communication to the Master Domain Manager is broken, because each machine understands its share in the schedules.

The Fault Tolerant Agent can be extended with specialized programs to access a certain environment, called Extended Agents. A sample Tivoli Workload Scheduler environment is shown in Figure 3-7.

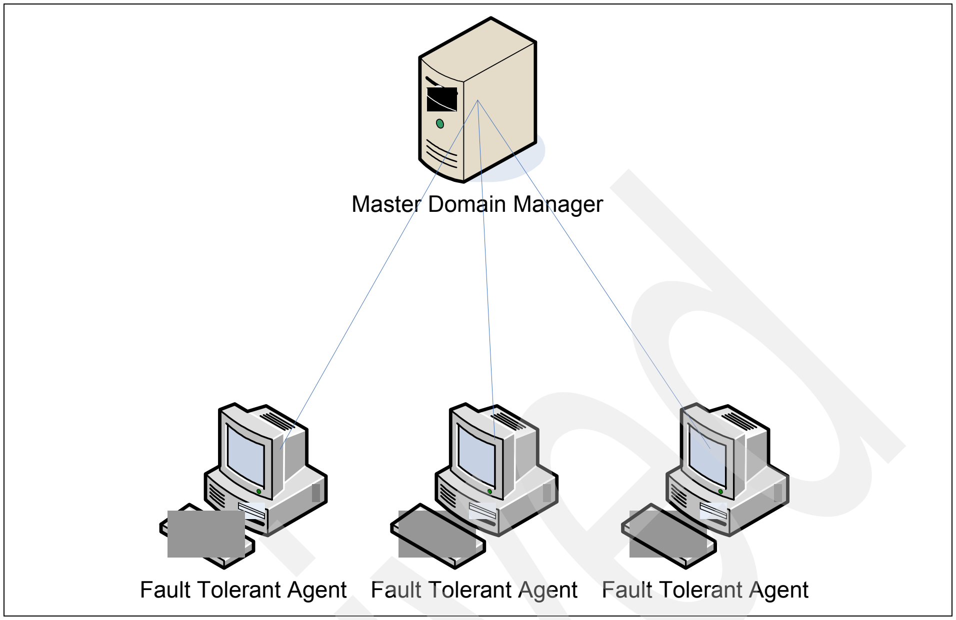


Figure 3-7 Tivoli Workload Scheduler

We do not discuss Tivoli Workload Scheduler concepts and implementation in this book. Refer to Tivoli Workload Scheduler product documentation for more information.

The Tivoli Workload Scheduler is managed using a Java-based Job Scheduling Console. After we install a Fault Tolerant Agent in the Tivoli Usage and Accounting Manager application server, we define it using the Job Scheduling Console as shown in Figure 3-8 on page 65.

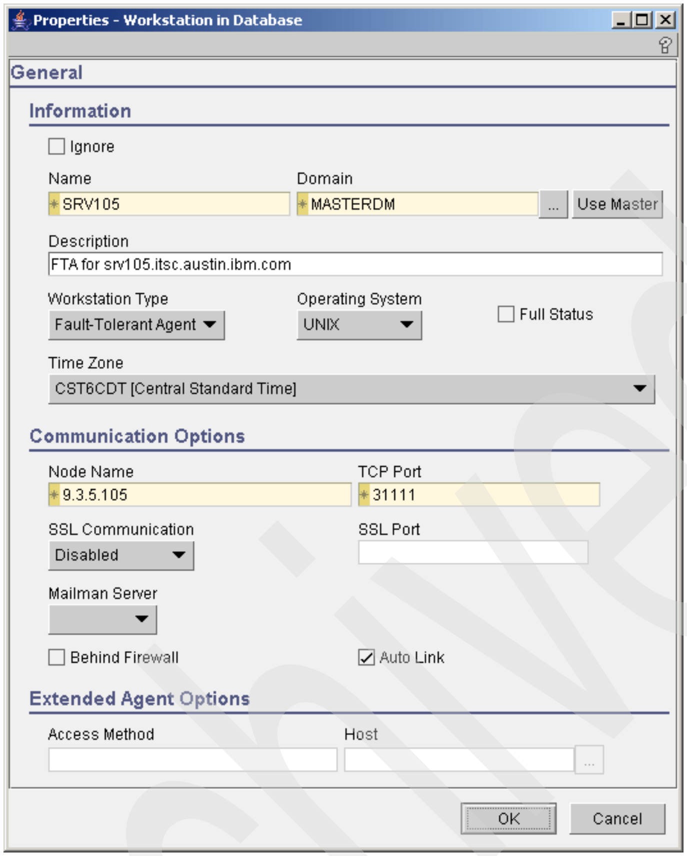


Figure 3-8 Defining the Fault Tolerant Agent

After the Fault Tolerant Agent is defined, we create an extended agent definition as shown in Figure 3-9 on page 66.

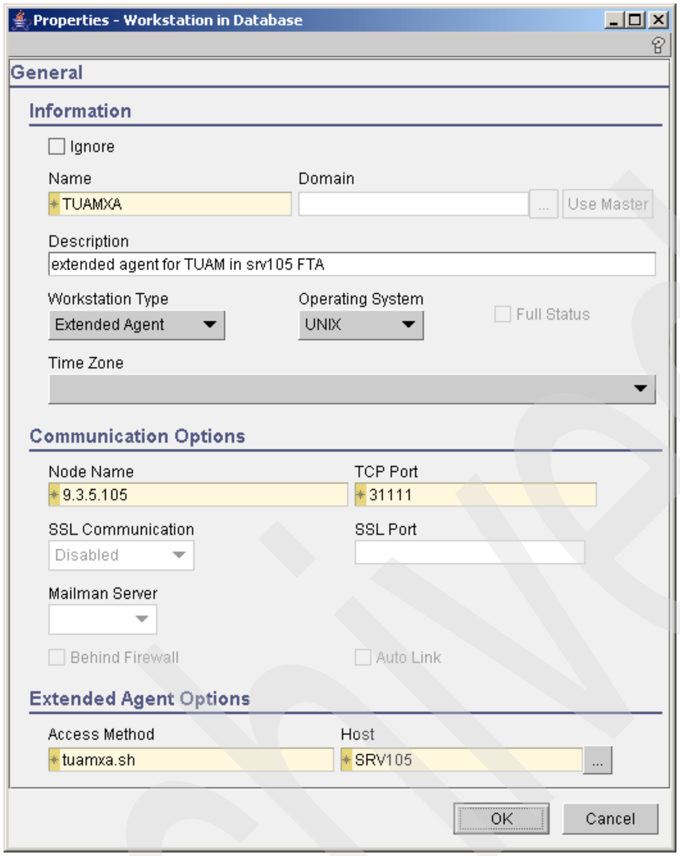


Figure 3-9 Defining the extended agent

In Figure 3-9, the extended agent is defined to reside within the SRV105 Fault Tolerant Agent and has an access method of tuamxa.sh. This means that the extended agent jobs are sent to the SRV105 Fault Tolerant Agent. In that machine, the FTA invokes the scripts access method (tuamxa.sh) from the methods sub directory of the Tivoli Workload Scheduler installation directory.

Once we have the workstations defined the plan has to be generated. The plan is a to do list for Tivoli Workload Scheduler that understand the jobs, workstations, and all their relationships. Typically a plan is generated every day on a schedule. A plan is generated using a program called JNextPlan.

After a new plan is generated that has our workstation definition, we check the view called All Defined Workstation as shown in Figure 3-10 on page 67.

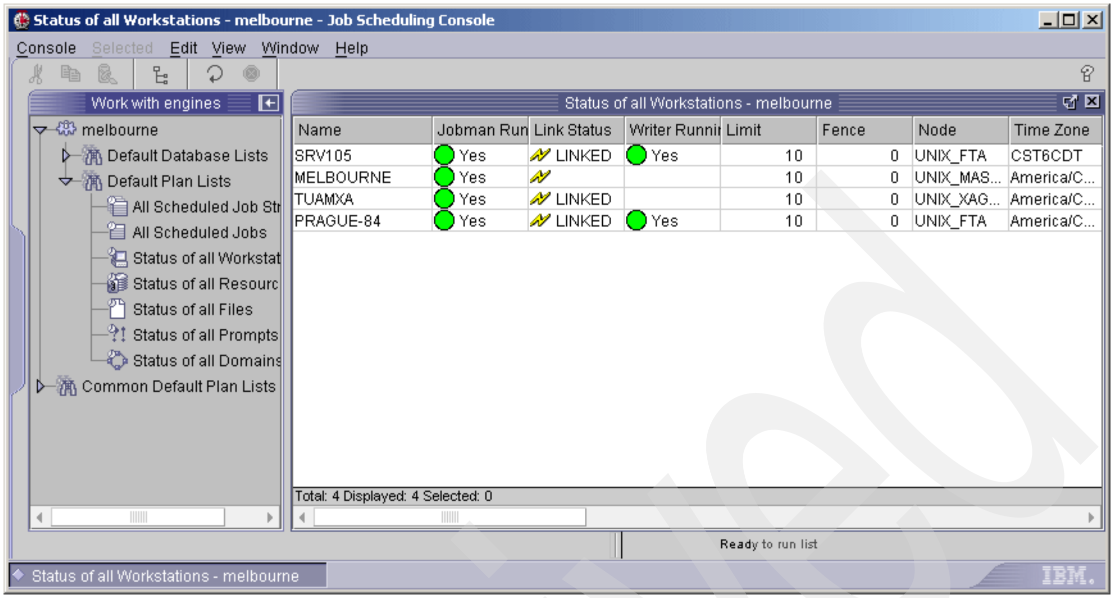
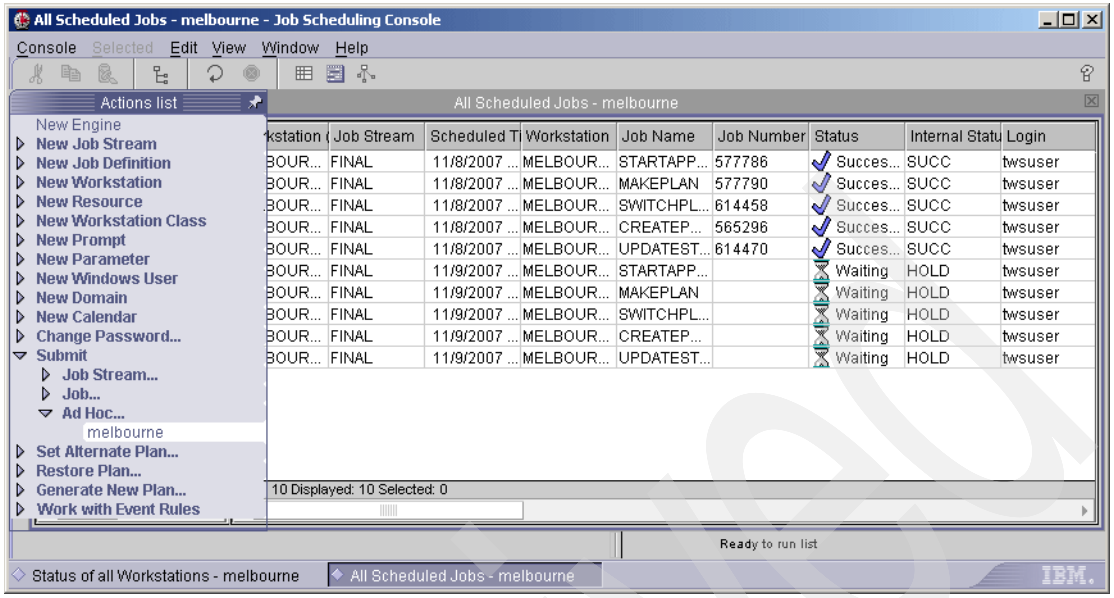


Figure 3-10 All defined workstation

Your Fault Tolerant Agent should be linked, with both Writer and Jobman running, while the extended agent would take the Fault Tolerant Agent's linked status.

To demonstrate how a job is run, use ad hoc submission of the job by selecting **Submit** → **Adhoc** → ***<scheduler name>***. See Figure 3-11 on page 68.



kstation	Job Stream	Scheduled Ti	Workstation	Job Name	Job Number	Status	Internal Statu	Login
BOUR...	FINAL	11/8/2007 ...	MELBOUR...	STARTAPP...	577786	Succes...	SUCC	twsuser
BOUR...	FINAL	11/8/2007 ...	MELBOUR...	MAKEPLAN	577790	Succes...	SUCC	twsuser
BOUR...	FINAL	11/8/2007 ...	MELBOUR...	SWITCHPL...	614458	Succes...	SUCC	twsuser
BOUR...	FINAL	11/8/2007 ...	MELBOUR...	CREATEP...	565296	Succes...	SUCC	twsuser
BOUR...	FINAL	11/8/2007 ...	MELBOUR...	UPDATEST...	614470	Succes...	SUCC	twsuser
BOUR...	FINAL	11/9/2007 ...	MELBOUR...	STARTAPP...		Waiting	HOLD	twsuser
BOUR...	FINAL	11/9/2007 ...	MELBOUR...	MAKEPLAN		Waiting	HOLD	twsuser
BOUR...	FINAL	11/9/2007 ...	MELBOUR...	SWITCHPL...		Waiting	HOLD	twsuser
BOUR...	FINAL	11/9/2007 ...	MELBOUR...	CREATEP...		Waiting	HOLD	twsuser
BOUR...	FINAL	11/9/2007 ...	MELBOUR...	UPDATEST...		Waiting	HOLD	twsuser

Figure 3-11 Submit adhoc job

Fill in your job in the prompt. As shown in Figure 3-12 on page 69, the job is to be run as the Tivoli Usage and Accounting Manager installation user, and runs on the extended agent.

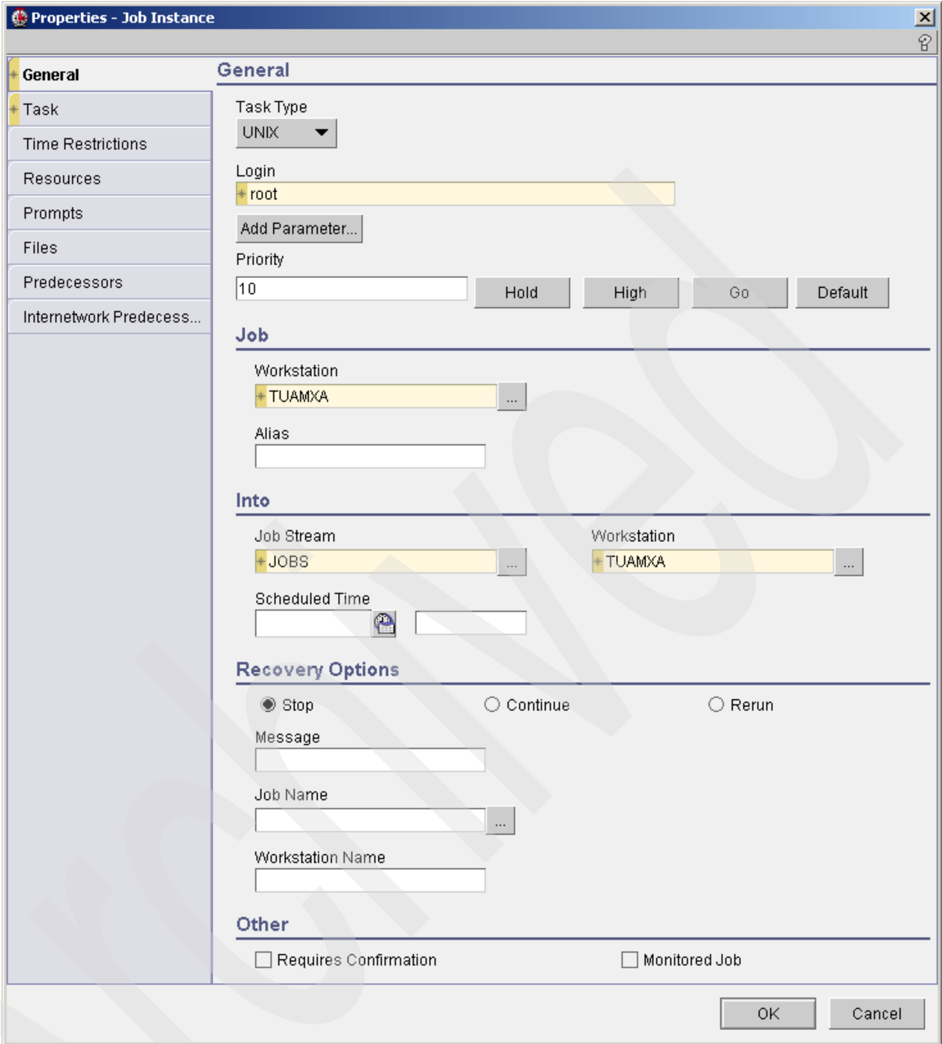


Figure 3-12 Job general definition

The argument for the job is defined in the Task tab as shown in Figure 3-13 on page 70. We used only the XML file that defines the job. Additional options such as job filter and run date can also be defined.

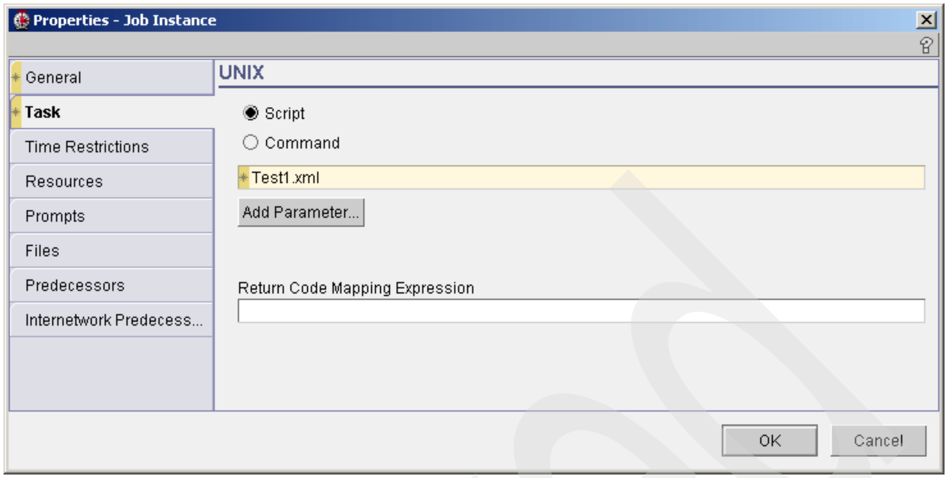


Figure 3-13 Defining the job task

When the job runs, it invokes our extended agent method, tuamxa.sh. An excerpt of the listing is shown in Example 3-7.

Example 3-7 The access method tuamxa.sh

```
#!/bin/sh
# *******************************************************************
#
# ********************************************************************

binDir="/opt/ibm/tuam/bin"
. /opt/ibm/tuam/bin/setupEnv.sh

TUAM_INSTALLPATH=$TUAM_HOME
JRELIB_PATH=$TUAM_HOME/ewas/java/jre/lib

# ***** Passed in parameter ******
ARGS=$*

# ***** TUAM Libraries **********
JR_CLASSPATH=$TUAM_LIB/tuamxa.jar:.:$CLASSPATH
JR_CLASSPATH=$JR_CLASSPATH:$TUAM_LIB/aucCommon.jar
   .  .  .

exec $JAVA_HOME/jre/bin/java -cp $JR_CLASSPATH
com.ibm.vbd.tuam.xa.TXAMethod $* 2>&1
```

The script is a modified startJobRunner script. Because it will run from the Fault Tolerant Agent, it has to be told about Tivoli Usage and Accounting Manager installation directory. It also invokes our custom method Java class called com.ibm.vbd.tuam.xa.TXAMethod. The full content of our tuamxa.bat is provided in “Sample script tuamxa.bat” on page 361and the listing of the Java class is provided in “TXAMethod Java class” on page 363. Note that the Job runner put the messages in the standard error, so we need to perform a redirection on the **`java`** command.

After the job executes, we can see the status of the job as shown in Figure 3-14.

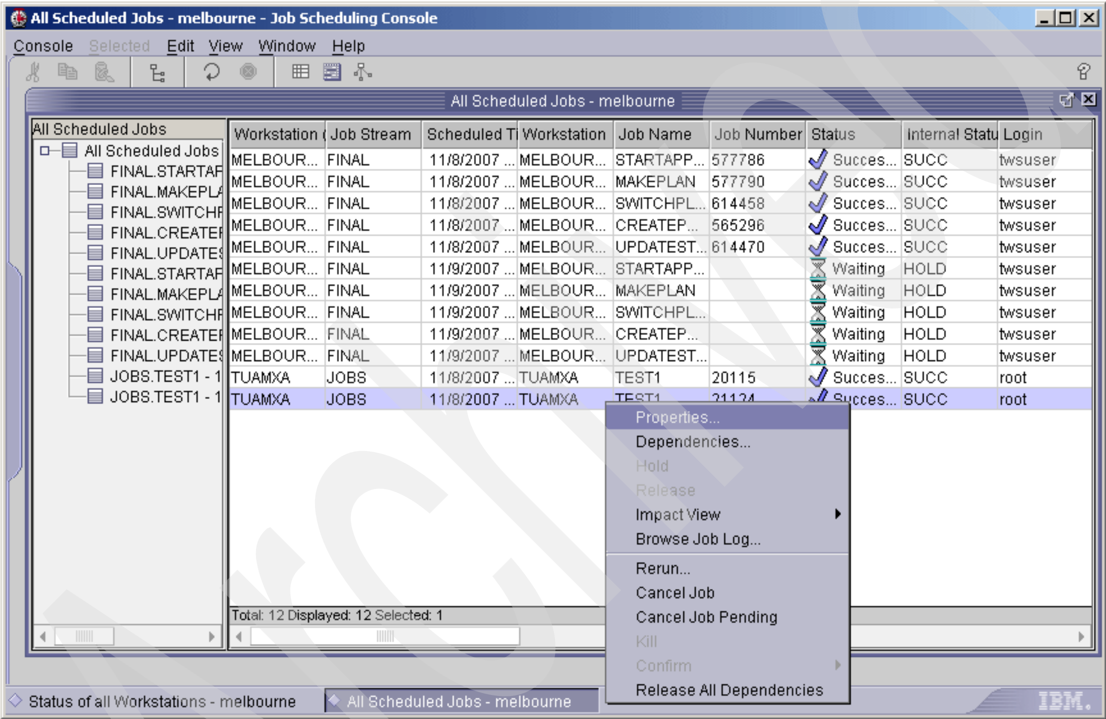


Figure 3-14 Job status

Right-click the successful job and select **Browse Job Log** to view the output of the job; our results are shown in Figure 3-15 on page 72.

Figure 3-15 Job output

4

Installation and configuration

This chapter describes installation and configuration of IBM Tivoli Usage and Accounting Manager. It covers the following topics:

4.1 Estimating database size

This section provides an overview of estimating the Tivoli Usage and Accounting Manager database growth. Because Tivoli Usage and Accounting Manager is a data collection and processing tools, it collects and loads data into the database and keeps it for some period of time. Estimating its growth is critical for ensuring that the space is properly allocated and the resulting performance impact can be addressed (such as the time to back up the data, query response time, replication needs, and so on).

Attention: The estimation technique described here has not been tested with an actual customer environment; it was used only for estimating database size in our sample environment. All estimation techniques are provided as is.

We start by checking our database size in our Windows directory or Linux filesystem just after it is being initialized. The data size is roughly 350 MB, including the database catalog and database log files.

4.1.1 Data elements

The primary growth in the database comes from the following usage and accounting data:

Resource utilization The collection of the resource usage metric from the AcctCSR file. Collection is provided by identifier for each resource (rate code). This is an optional collection. You do not of need to collect the resource usage.

Billing summary This provides a summary usage for each resource (rate code) by account code. It is important that the input to the billing cycle be sorted by account code to minimize duplicate summary records. The data is one-to-one mapping from BillSummary.txt file.

Billing detail This provides individual entries from the AcctCSR file. It gives individual occurrences of source usage by resource name (rate code). This links to the identifier table for getting the identifier key for each of the entries here. The data is one-to-one mapping from BillDetail.txt file.

Identifier table This lists the identifiers that are used by each Billing detail entry. The data is one-to-one mapping from Ident.txt file.

An overview of the relationship between these tables is shown in Figure 4-1.

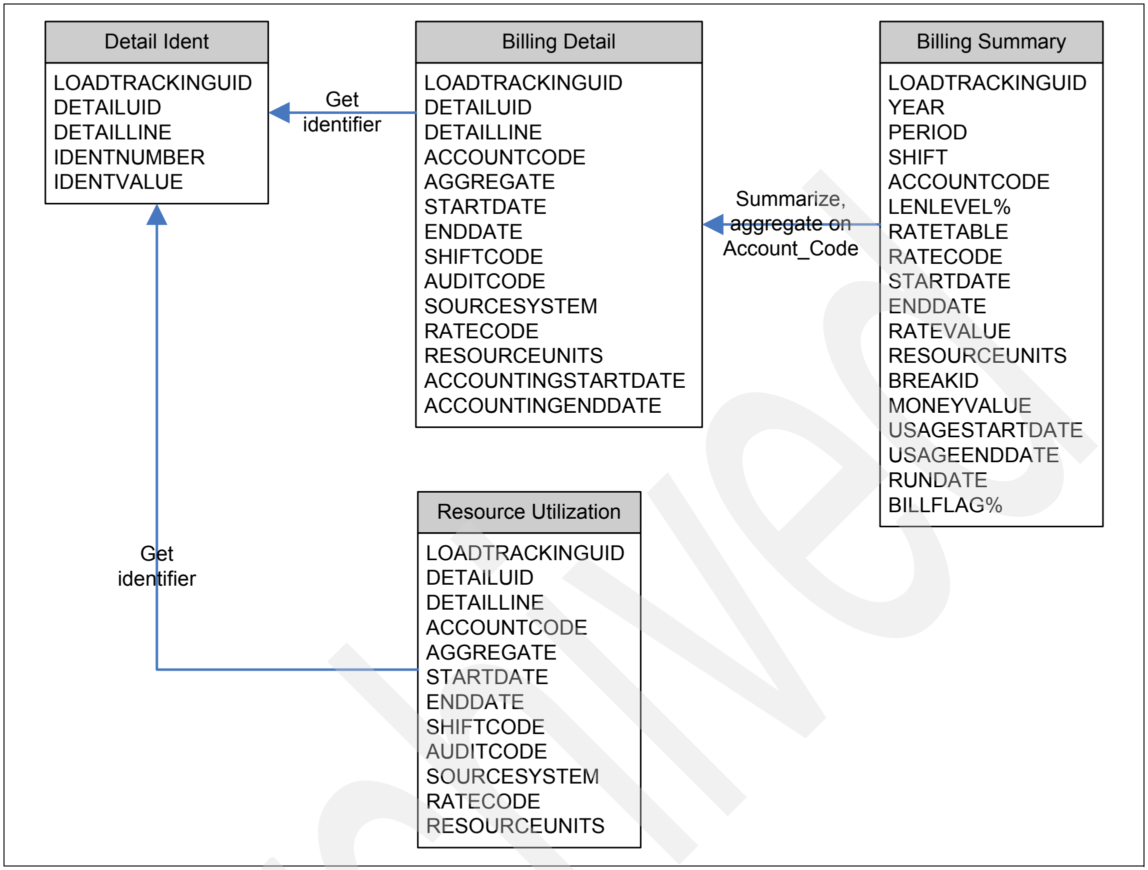


Figure 4-1 Table relationship

Some important tips to keep database size manageable:

- You should run the DBpurge program using Job runner to remove old data. Because Tivoli Usage and Accounting Manager data is an accounting financial tool, you may want to archive the data first.
- Sort by Account_Code field before running the billing or use a CSR+ file for Bill input.
- Only collect the identifiers and resources that you are interested in. Modify the sample collection jobs, change the mapping, and remove any unwanted identifiers and resources fields.

4.1.2 Growth factors

Now we look at the tables and analyze which parameters affect their sizes. The following are the size multipliers:

Number of days The retention period of your data before you run the purge step to remove it.

Number of shifts Number of shifts in a day that need different rate codes.

Collection source Each collection source is processed with different jobs. Each will generate a different set of data.

Account code All billing and resource tables are indexed by the Account code entry. This is the primary retrieval mechanism for Tivoli Usage and Accounting Manager data. You must estimate the number of distinct account codes.

Number of resources The resources are mapped directly as rate codes. These rate codes are the secondary search mechanism for Tivoli Usage and Accounting Manager.

Number of identifiers Each identifier is put in a different row in the CIMSDETAILIDENT table.

Identifier mix This is the number of unique identifiers in each collection. You must be able to estimate this number by your understanding of the collection process. As an example, for Windows processes, you can count the number of running processes within the day as the identifier mix.

Regarding the tables themselves, which of the identified items maps? Table 4-1 lists the factors that have a significant impact and the estimated row size of the tables.

Table 4-1 Table estimation

Name	Row size[a]	Affecting source
CIMSRESOURCE UTILIZATION	300	Source, Account_Code, Identifier mix, RateCode, Shift, #days
CIMSSUMMARY	300	Source, Account_Code, RateCode, Shift, #days
CIMSDETAIL	350	Source, Account_Code, Identifier mix, Rate per id, Shift, #days
CIMSDETAILIDENT	75	<Ident mix> x <Ident count>

a. The row size is an estimate based on the table structure and using the assumption that a VARCHAR or VARGRAPHIC columns use half its capacity.

4.1.3 Sample growth estimate

For the purpose of this sample, the pertinent facts are as follows:

- Data is kept for two years, except the detail data, which is kept for one year only.
- There are two shifts collected.
- The account structure is in the form of: client - department - application - host.
- Usage information is collected for UNIX processes and Windows processes only.
- The average identifier length is 20 characters.
- Audit codes are not used.
- Percentage of completely filled records is 75% because some of the accounting data is only partially filled. Some of the metrics may not appear in all records.

For the UNIX processes, collection is performed on 15 machines and 12 resource metrics are collected. The identifier fields are Feed, Account_Code, hostname, userName, and process. The estimated number of processes per day is 250.

For Windows processes, collection is performed on 20 machines and 8 resource metrics are collected. The identifier fields are Feed, Account_Code, Server, User, processName (we assume that BasePriority, PriorityClass, and ProgramPath fields are dropped). The estimated number of process per day is 100.

The number of unique identifiers in both UNIX process and Windows process is the estimated number of processes.

The number of account codes is then derived from its structure. As mentioned previously, the account code structure is client - department - application - host. It is important to plan this structure first, including how these items can be identified. This example assumes that the account code elements are retrieved as follows:

- Host is retrieved from hostname or server_id identifier.
- Application is derived using a lookup table based on the server, user, and program name.
- Department is derived from application.
- Client is derived from department.

Based on the specification, we conclude that the number of unique account codes will be the same as the number of applications (or applications by host).

Now we can start performing the calculation. First, we collect the multipliers as shown in Figure 4-2.

Account structure worksheet			
#client	#dept	#appl	#host
4	12	42	15
4	12	30	20

source feed	Number of identifier fields	Number of resource fields	Number of unique identifier combinations	Number of source data	Number of unique account codes
UNIX	5	12	250	15	42
Windows	5	8	100	20	30

# shift	2
Default # days	730
Average identifier length	20
Length of Audit code	0
Pct of records with all ident	75%

Figure 4-2 Estimating the multipliers

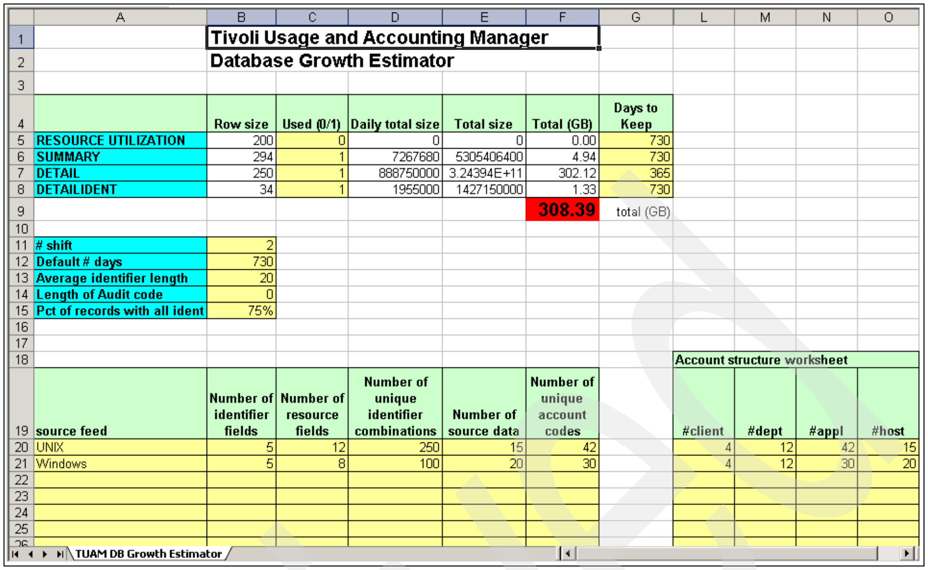
In Figure 4-2, the account structure is estimated by listing the component occurrences. We have decided to use the number of applications as the number of unique account codes. All the other numbers are collected from the discussion.

The resulting table sizes are shown in Figure 4-3.

	Row size	Used (0/1)	Daily total size	Total size	Total (GB)	Days to Keep
RESOURCE UTILIZATION	200	0	0	0	0.00	730
SUMMARY	294	1	7267680	5305406400	4.94	730
DETAIL	250	1	888750000	3.24394E+11	302.12	365
DETAILIDENT	34	1	1955000	1427150000	1.33	730
					308.39	total (GB)

Figure 4-3 Table sizes result

As shown in this figure, the total data size is around 309 GB. We assume that we do not collect the resource utilization table. We performed the calculation using the single spreadsheet shown in Figure 4-4 on page 79.


Tivoli Usage and Accounting Manager
Database Growth Estimator

	Row size	Used (0/1)	Daily total size	Total size	Total (GB)	Days to Keep
RESOURCE UTILIZATION	200	0	0	0	0.00	730
SUMMARY	294	1	7267680	5305406400	4.94	730
DETAIL	250	1	888750000	3.24394E+11	302.12	365
DETAILIDENT	34	1	1955000	1427150000	1.33	730
					308.39	total (GB)

# shift	2
Default # days	730
Average identifier length	20
Length of Audit code	0
Pct of records with all ident	75%

source feed	Number of identifier fields	Number of resource fields	Number of unique identifier combinations	Number of source data	Number of unique account codes	Account structure worksheet #client	#dept	#appl	#host
UNIX	5	12	250	15	42	4	12	42	15
Windows	5	8	100	20	30	4	12	30	20

TUAM DB Growth Estimator

Figure 4-4 Estimator

The file is provided in the additional material.

4.2 Architectural considerations

Tivoli Usage and Accounting Manager is comprised of three main components: the administration application, the reporting component, and the processing component. All these components connect to a data source to store and retrieve information. There are several possible configurations that you can use. You should evaluate your requirements and resource availability to determine the best architecture for your environment.

Depending on the platform selected for each of the components, there are minimum hardware requirements and prerequisite software. For a complete list of requirements we recommend that you review the installation instructions in the Tivoli Usage and Accounting Manager documentation available at:

```
http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?topic=
/com.ibm.ituam.doc_7.1/welcome.htm
```

4.2.1 Components

This section discusses the Tivoli Usage and Accounting Manager components.

- Administration server

  The administration server is based on the Integrated Solutions Console (ISC). ISC is an administration tool that runs inside an embedded WebSphere Application Server. The administration server component can execute on Windows 2003 Server, and various versions of Linux or UNIX.

- Processing server

  The data processing server is typically run on the same machine as the administration server. However, there may be some requirement, such as load balancing or platform considerations (some processing that is supported to run on a Windows-based platform only), that would preclude this. The Tivoli Usage and Accounting Manager processing engine applies business rules to data and stores information in a database. This component can run on Windows and various versions of Linux or UNIX.

  There are two kinds of data collectors:

  a. Data collectors deployed to machines that you want to collect data from and that do not have the capability to collect the data on their own.

  b. Tivoli Usage and Accounting Manager data collectors that extract usage data from OS or application provided accounting data. These data collectors can be deployed on the processing server directly.

- Reporting server

  The Tivoli Usage and Accounting Manager reporting server component currently is only supported on Windows 2003 Server and also requires:

  – The Report Viewer for Microsoft SQL Server Reporting Services

  – Microsoft Internet Information Services (IIS)

  If you require a non-Windows based reporting server, there is a limited report function provided that is based on Business Intelligence and Reporting Tools (BIRT) open source platform.

- Database server

  The database server stores the information produced by the processing engine. It does not have to run on the same system or platform as the processing engine. The following database technologies are supported by Tivoli Usage and Accounting Manager:

  – DB2 Universal Database™ (UDB) for Linux, UNIX, and Windows Version 8.1, 8.2, or 9.1

  – DB2 UDB for z/OS Version 8.1 and later

  – Microsoft SQL Server 2000 or 2005 with the latest service pack

  – Oracle 9 or 10 for UNIX or Windows

> **Note:** If SQL Server is used as the database server, then the JDBC driver for SQL Server 2005 should always be used.

All components need access to the appropriate JDBC drivers for the database you choose to use.

4.2.2 Sample configuration

Figure 4-5 shows the configuration that we used to prepare this chapter. We describe how to install the enterprise edition of the software as follows:

- The Tivoli Usage and Accounting Manager administration and processing servers, including the primary database, are installed on a Red Hat Enterprise Linux AS release 4 machine.
- The reporting server is installed on a Windows 2003 Server machine. The installation includes a processing engine that can run Windows-based processing.

We call this a multiple server installation.

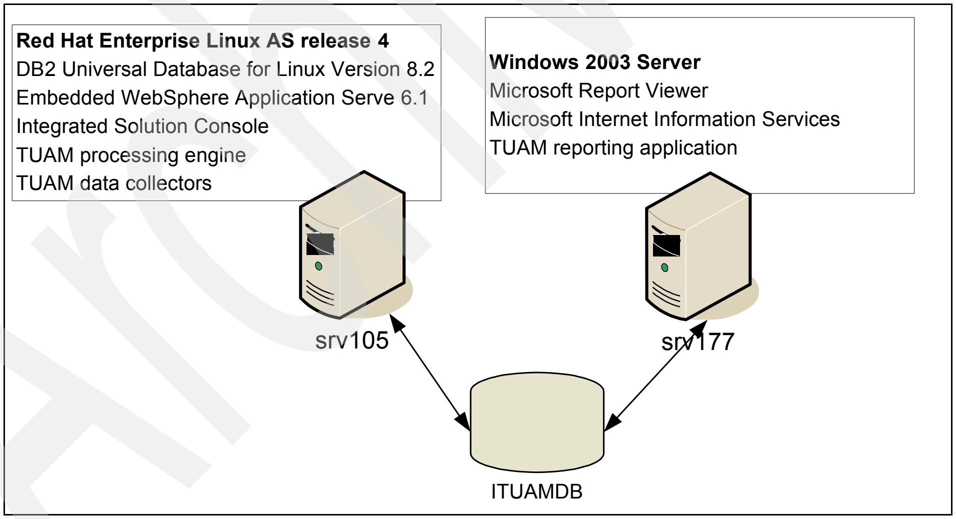


Figure 4-5 Initial configuration of the environment

4.2.3 Installation process overview

At a high level, the installation process consists of these steps:

1. Install the administration and processing server in srv105 machine. This machine is similar to the application server that is mentioned in the Tivoli Usage and Accounting Manager product manual. Details are provided in 4.3, "Installing the administration server" on page 82 and 4.4, "Verifying the installation of the Application server" on page 87.
2. Create the database and perform initial configuration and database initialization. Details are provided in 4.5, "Initial configuration of the Application server" on page 89 and 4.6, "Verifying the database configuration" on page 101.
3. Install the reporting server as explained in 4.7, "Installing and configuring the reporting server" on page 103 and 4.8, "Verifying the installation of the Report server" on page 113.
4. Install data collection in the target systems. This is discussed along with the scenarios. The basic installation method for Windows is to execute the installation wizard, while for UNIX/Linux we use the sample deployment job.

4.3 Installing the administration server

Prior to installation of the administration server, verify that:

- You have all the required hardware resource and software prerequisites installed on the application server platform
- You are signed on to the administration server with the root account or Windows Administrator

We install the administration server using a graphical user interface (GUI) installation. For Linux, this graphical installation mode is invoked using an X-windows session and the following steps:

1. Make the install files accessible by mounting a file system that contains the install files.
2. Ensure that the required files are located in the same directory as the installation file. For our installation on Linux, we placed following files in the same directory as the setup-tuam-ee-7-1-0-linux_ia32.bin file:
   - EmbeddedExpress_linux_ia32.zip
   - ISCAE71_4_EWASv61.zip
   - setup-tuam-wpc-7-1-0-windows_32_64.exe

3. Set the DISPLAY environment variable on the application server to identify the X-windows server that you are using for the GUI mode installation. We issued the command:

```
export DISPLAY=PC_3C00B:0.0
```

The actual installation steps are performed from the administration server and are as follows:

1. On the server, change to the directory that contains the executables for installation:

```
cd /ti7b55/noback/200709290755/ismp/ee/
```

2. Run the installation program `./setup-tuam-ee-7-1-0-linux_ia32.bin`. The installation wizard starts on the client machine (PC_3C00B) and the welcome window is displayed, as shown in Figure 4-6.

Figure 4-6 Installation welcome

3. Accept the license agreement as shown in Figure 4-7 on page 84.

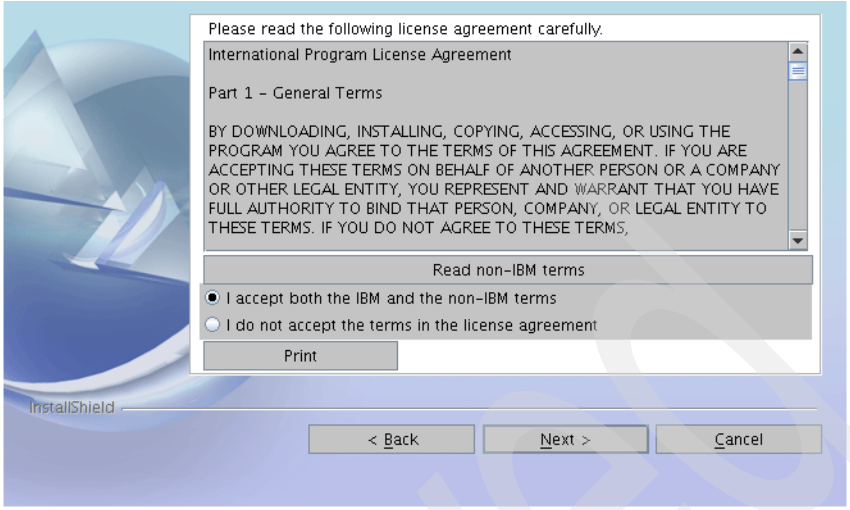


Figure 4-7 License agreement acceptance

4. Install Tivoli Usage and Accounting Manager into the /opt/ibm/tuam directory, which is the default directory (Figure 4-8).

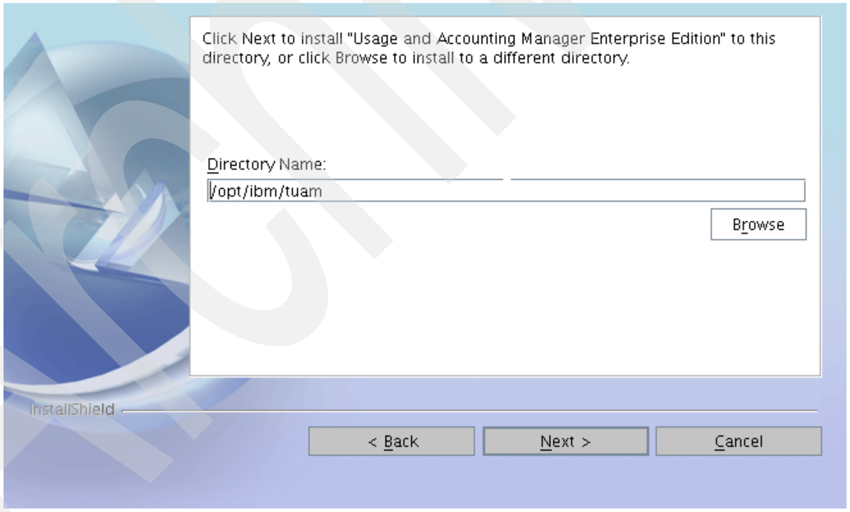


Figure 4-8 Installation path for the Application server

5. Figure 4-9 shows the installation summary window containing the expected disk footprint.

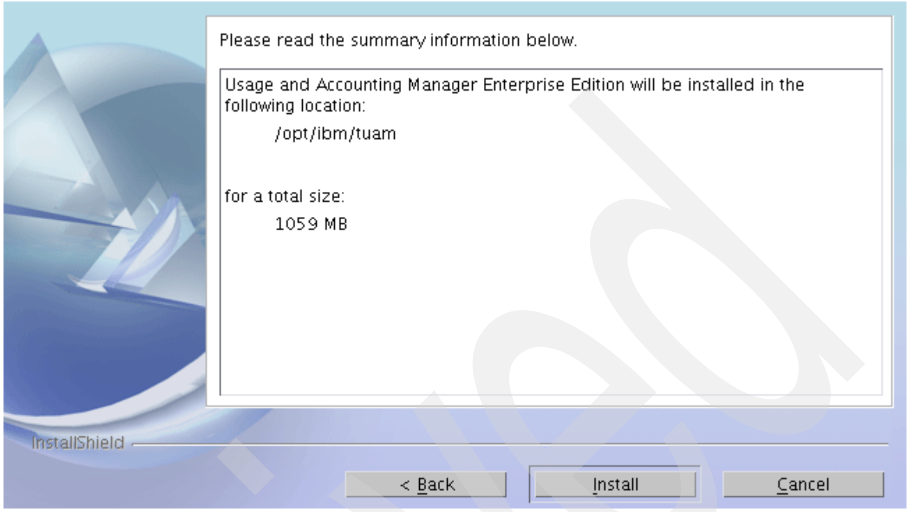


Figure 4-9 Disk space footprint

6. The installation proceeds, with progress indicated by a status bar. Successful installation of the server is indicated as shown in Figure 4-10.

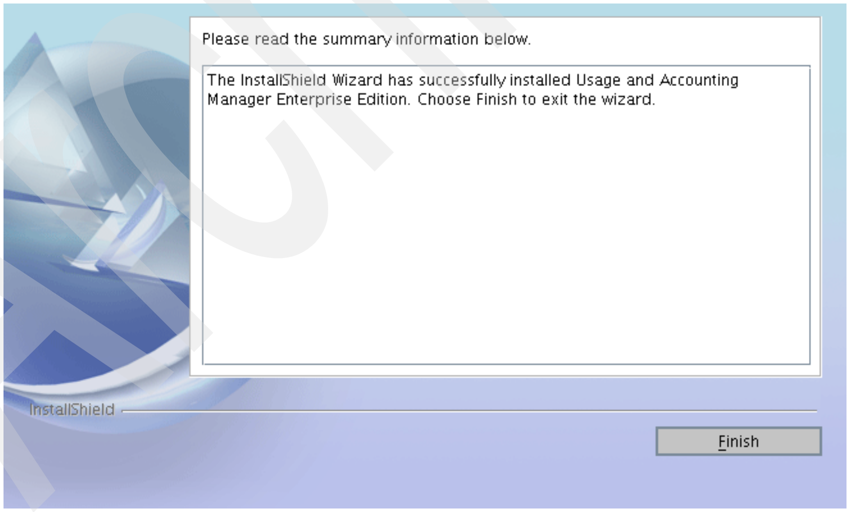


Figure 4-10 Successful completion of the installation

During this process, it is possible that installation is not completed successfully. Figure 4-11 on page 86 shows an error message when the installation of the

Application server is not successful, If your installation fails, review the log file to determine what caused the failure. If you need to uninstall the Application server, the uninstall program is located in the _uninst directory off the main installation directory. The uninstall program is called `uninstaller.bin`.

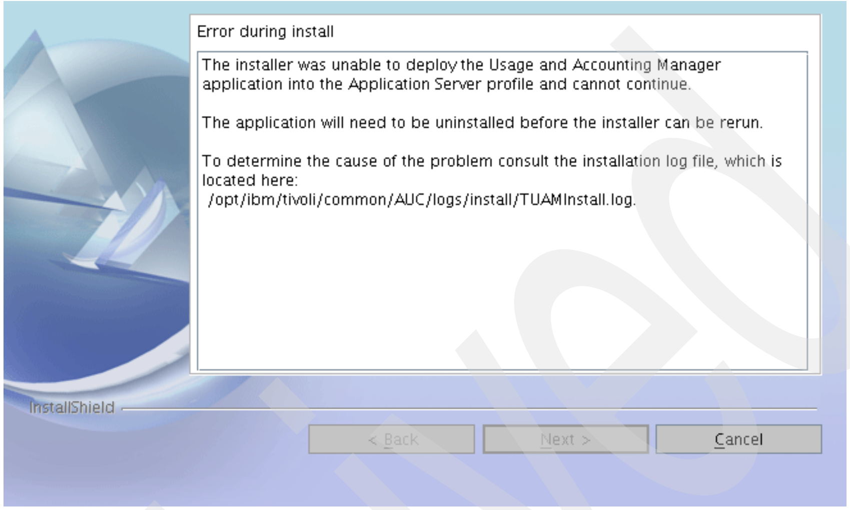


Figure 4-11 Error during installation

Figure 4-12 on page 87 shows a warning message when the installation of the Application server is not successful, This warning message appears because the setup-tuam-wpc-7-1-0-windows_32_64.exe file is not in the same directory as the Tivoli Usage and Accounting Manager installer program. If you receive this message, manually copy the setup-tuam-wpc-7-1-0-windows_32_64.exe file into the directory specified in the message.

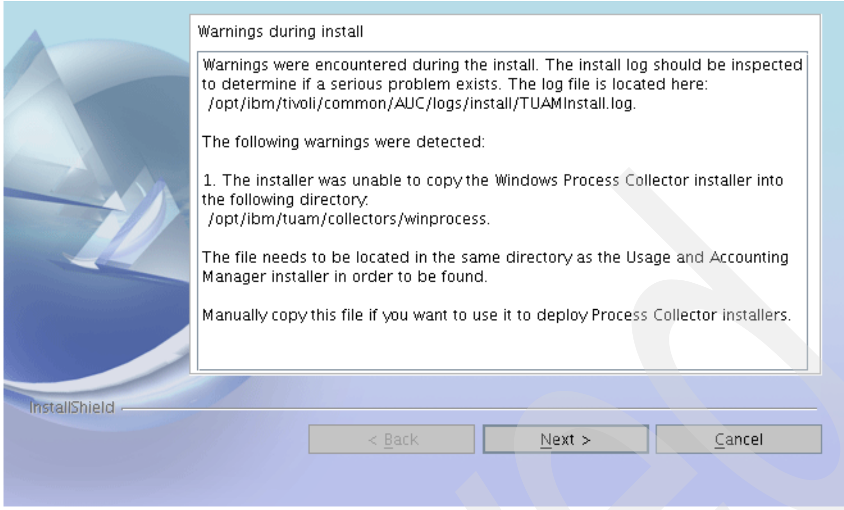


Figure 4-12 Warning message

Note: As shown in Figure 4-11 and Figure 4-12, the installation log file is not in the directory for the Tivoli Usage and Accounting Manager installation. In our case, the Tivoli Usage and Accounting Manager directory is /opt/ibm/tuam, and the installation log file is in /opt/ibm/tivoli/common/AUC/logs/install.

4.4 Verifying the installation of the Application server

To verify that the administration server has been installed, run the Tivoli Usage and Accounting Manager console application as follows:

1. Using the session for the administration server system, change to the embedded WebSphere Application Server directory in /opt/ibm/tuam/ewas/profiles/AppSrv01/bin.
2. From this directory, start the embedded WebSphere Application Server server using the startServer.sh script. Figure 4-13 on page 88 shows the messages displayed by the application as it starts.

```
[root@srv105 bin]# ./startServer.sh server1
ADMU0116I: Tool information is being logged in file
/opt/ibm/tuam/ewas/profiles/AppSrv01/logs/server1/startServer.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is 29586
```

Figure 4-13 Starting ISC

3. Open a Web browser and direct it to the Integrated Solution Console URL:

   `http://srv105:11052/ibm/console`

4. The welcome page is displayed (Figure 4-14). Verify that the product information matches the version that you have installed.

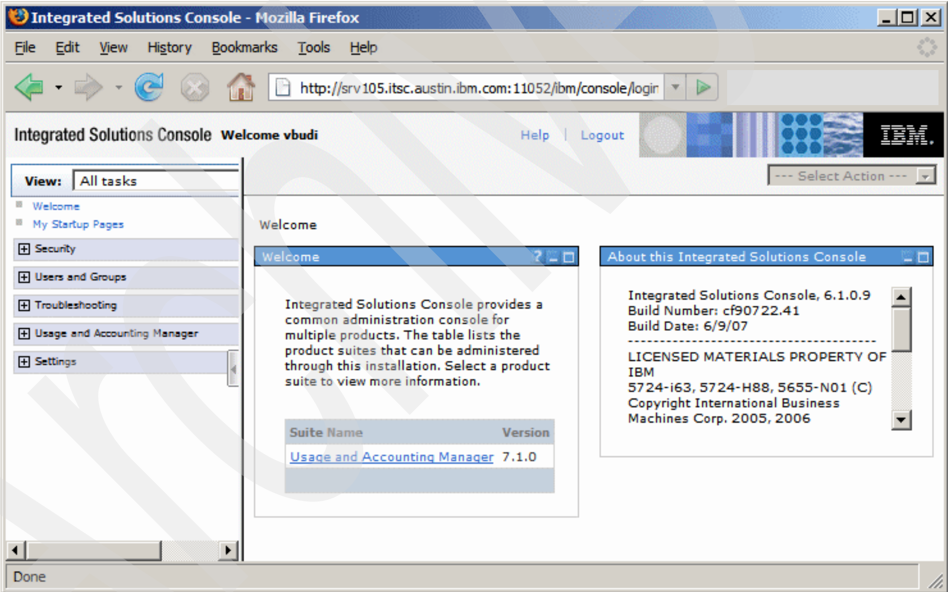


Figure 4-14 Verifying the installation of the Application server

4.5 Initial configuration of the Application server

The detailed steps used to perform the initial configuration and testing of Tivoli Usage and Accounting Manager are presented in this section. As an overview, the steps are:

- 4.5.1, “Database creation” on page 89
- 4.5.2, “Configure JDBC driver” on page 91
- 4.5.3, “Configure the data sources” on page 92
- 4.5.4, “Initialize the Tivoli Usage and Accounting Manager database” on page 95
- 4.5.5, “Run the samples for testing the installation” on page 97
- 4.5.6, “Set up the processing directories” on page 100

4.5.1 Database creation

Tivoli Usage and Accounting Manager requires the use of a database for its administration, processing, and reporting. In our sample environment, the database resides in the administration server. We are using DB2 Universal Database V8.2 for our database. We use the standard installation process for installing DB2 Universal Database V8.2 Enterprise Server Edition.

Because the machine is a Linux machine, the default DB2 instance with the name of db2inst1 is created by the graphical installation wizard. We create a database called ITUAMDB as shown in Figure 4-15. The database has to be defined with a UNICODE codepage, such as UTF-8.

```
[root@srv105 ~]# su - db2inst1
[db2inst1@srv105]$ db2 create db ITUAMDB using codeset UTF-8 territory
US
DB20000I  The CREATE DATABASE command completed successfully.
```

Figure 4-15 Database creation

Tivoli Usage and Accounting Manager requires a database with page size of at least 16K. This is not defined in the default database; perform the following steps to enable it:

1. Log on to the system containing the DB2 UDB database as the database user, db2inst1. (In a Windows machine, you must start the DB2 command processor.) Query the DB2 system tables to determine whether a large enough buffer pool exists. The SQL statement and results are in Figure 4-16

```
db2 => SELECT PAGESIZE, SUBSTR(BPNAME,1,20) AS BUFFERPOOL FROM
SYSCAT.BUFFERPOOLS

PAGESIZE    BUFFERPOOL
----------- --------------------
       4096 IBMDEFAULTBP
```

Figure 4-16 SQL query to determine if a large enough buffer pool exists

2. Define a buffer pool with a page size larger than the 4K default using the CREATE BUFFERPOOL command. The SQL syntax is shown in Figure 4-17.

```
db2 => create bufferpool BP32K size 1000 pagesize 32k
SQL20189W  The buffer pool operation (CREATE/ALTER) will not take
effect until the next database startup due to insufficient
memory. SQLSTATE=01657
```

Figure 4-17 Creation of a larger buffer pool

Because of the SQLSTATE of 01657, we must stop and restart the database manager, but we decided to create the user tablespace and temporary tablespace that use this buffer pool first.

3. Create a regular tablespace that uses the larger buffer pool as shown in Figure 4-18.

```
db2 => create tablespace USERSPACE2 PAGESIZE 32K managed by system
using ('/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLT0004')
bufferpool BP32K
DB20000I  The SQL command completed successfully.
```

Figure 4-18 Creation of a regular tablespace

4. Create a temporary tablespace that uses the larger buffer pool as shown in Figure 4-19. Ensure that the temporary parameter is used when creating this tablespace.

```
db2 => create temporary tablespace TEMPSPACE2 PAGESIZE 32K managed
by system using
('/home/db2inst1/db2inst1/NODE0000/SQL00002/SQLT0005') bufferpool
BP32K
DB20000I  The SQL command completed successfully.
```

Figure 4-19 Creation of a temporary tablespace

5. Restart the database instance. Use the LIST APPLICATION command to check whether there are any processes that are still using the database. The restart process is shown in Figure 4-20.

```
[db2inst1@srv105 ~]$ db2 list application
SQL1611W  No data was returned by Database System Monitor.
SQLSTATE=00000
[db2inst1@srv105 ~]$ db2stop
10/25/2007 08:41:55     0   0   SQL1064N  DB2STOP processing was
successful.
SQL1064N  DB2STOP processing was successful.
[db2inst1@srv105 ~]$ db2start
10/25/2007 08:42:00     0   0   SQL1063N  DB2START processing was
successful.
SQL1063N  DB2START processing was successful.
```

Figure 4-20 Restarting the database manager

4.5.2 Configure JDBC driver

The configuration of the JDBC driver for Tivoli Usage and Accounting Manager depends on the database software that has been installed. The DB2 Universal Database that we used came with the JDBC drivers. The default path for DB2 JDBC driver in Linux system is under /opt/IBM/db2/V8.1/java. We used the db2jcc.jar and db2jcc_license_cu.jar files for the JDBC driver.

To configure Tivoli Usage and Accounting Manager to use the JDBC driver, go to the ISC menu and select **Usage and Accounting Manager** → **System Maintenance** → **Configuration**. In the Driver tab, click **New** to define the driver. Enter the path and filename of the DB2 UDB driver as shown in Figure 4-21 on page 92. Click **OK** when done.

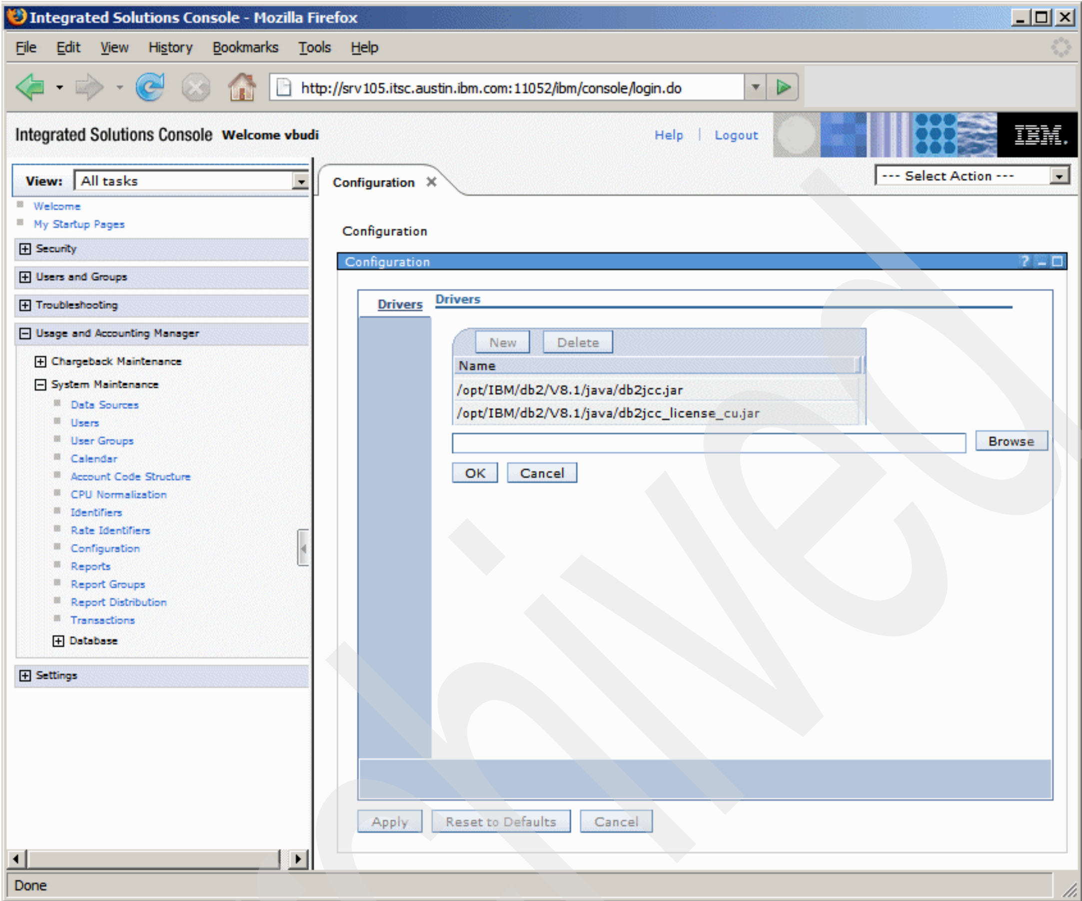


Figure 4-21 Configuring the Tivoli Usage and Accounting Manager JDBC driver

> **Note:** If you are configuring the JDBC drivers for DB2 UDB, ensure that you add both the db2jcc.jar and the license jar file as JDBC drivers.

4.5.3 Configure the data sources

Using the ISC, add the following two types of database sources:

- Server data source, which is the repository for Tivoli Usage and Accounting Manager
- Collector data source, which is the database that you collect data from

Perform the following steps to add Tivoli Usage and Accounting Manager database as a Server data source:

1. From the ISC menu select **Usage and Accounting Manager** → **System Maintenance** → **Data Sources**.
2. In the Configuration window select **Server**, then click **New**.
3. Enter the details about the DB2 UDB database as shown in Figure 4-22.

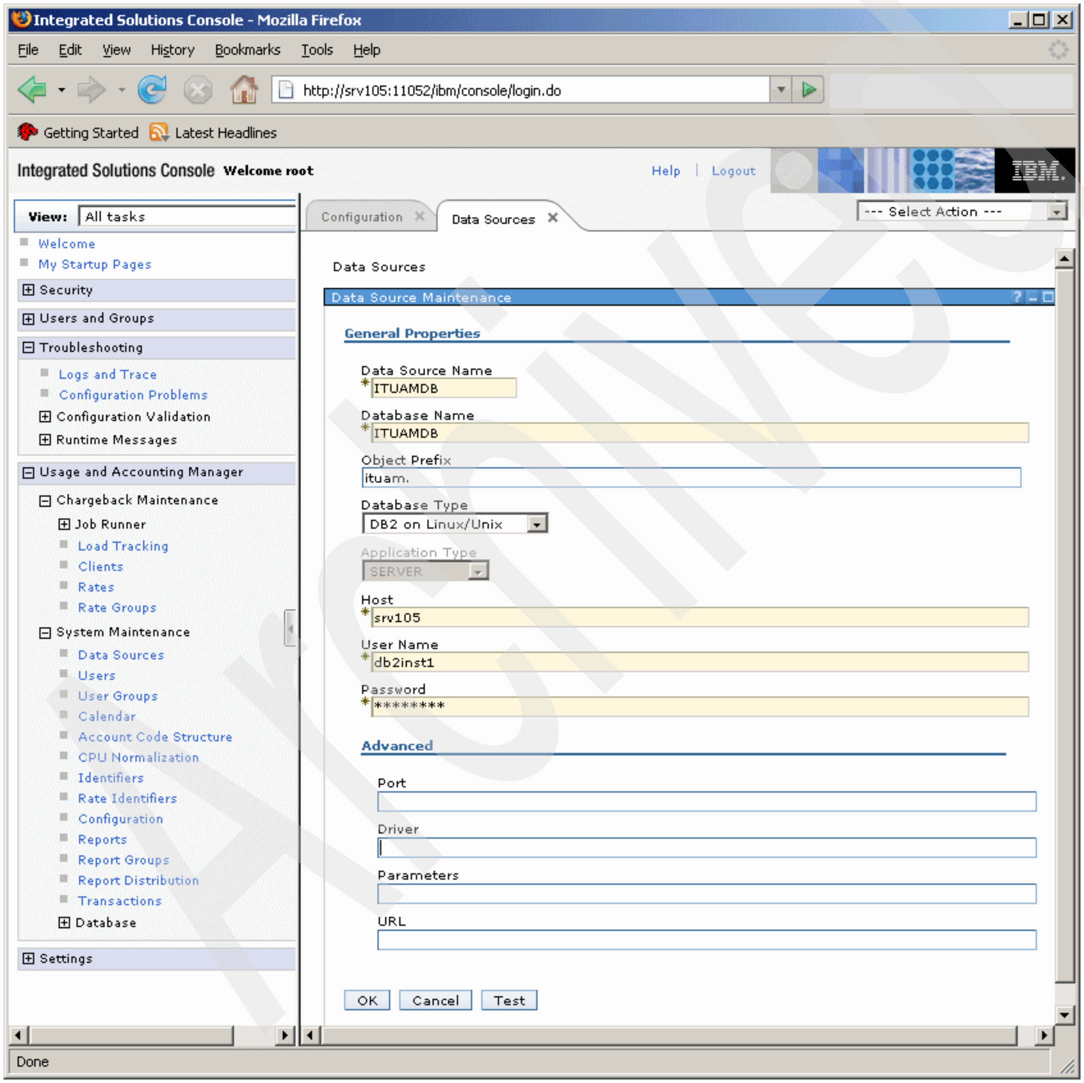


Figure 4-22 Define the Server data source

4. Verify that the connection to the Server database is operational by selecting **Test**. If the test of the database connection is successful, an information message displays `Connection was successful`. If the connection fails, messages are written to log files. The log files are located in the /opt/ibm/tuam/ewas/profiles/AppSrv01/logs/server1 directory and the /opt/ibm/tuam/logs/server directory.

   For example, if the database has not been created you receive the message shown in Figure 4-23.

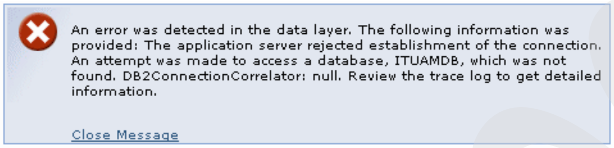


Figure 4-23 Error when database does not exist

   The SystemOut.log contains the information shown in Figure 4-24. This log file is in the /opt/ibm/tuam/ewas/profiles/AppSrv01/logs/server1 directory.

```
[10/3/07 10:27:14:919 CDT] 00000053 ITUAMLogger   E
com.ibm.tivoli.tuam.dataaccess.persistence.dao.jdbc.DataSourceManage
rDBJdbcDao isConnectionValid CANNOT_EXECUTE_SQLSTATEMENT_ERROR
[10/3/07 10:30:01:324 CDT] 00000054 SystemOut     O constructing an
AcctMgrDataAccessException with message: The application server
rejected establishment of the connection.  An attempt was made to
access a database, ITUAMDB, which was not found.
DB2ConnectionCorrelator: null and cause:
com.ibm.db2.jcc.a.DisconnectException: The application server
rejected establishment of the connection.  An attempt was made to
access a database, ITUAMDB, which was not found.
DB2ConnectionCorrelator: null
[10/3/07 10:30:01:324 CDT] 00000054 SystemOut     O    - cause is
not null: com.ibm.db2.jcc.a.DisconnectException: The application
server rejected establishment of the connection.  An attempt was
made to access a database, ITUAMDB, which was not found.
DB2ConnectionCorrelator: null
```

Figure 4-24 Error messages in the trace file

5. We used this single Tivoli Usage and Accounting Manager database for administration, processing, and reporting. To define it as such, from the ISC menu, select **Usage and Accounting Manager** → **System Maintenance** → **Data Sources** → **Server**. The Data Sources are listed.

6. Use the **View** pop-up menu for the Tivoli Usage and Accounting Manager database as shown in Figure 4-25. Select **Set Admin**, **Set Processing** and **Set Report**.

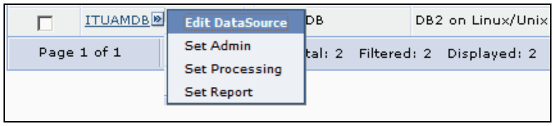


Figure 4-25 Setting the default Data Source

7. The Default Admin, Default Processing, and Default Reporting columns for the Tivoli Usage and Accounting Manager database should now be set to `Yes` as in Figure 4-26.

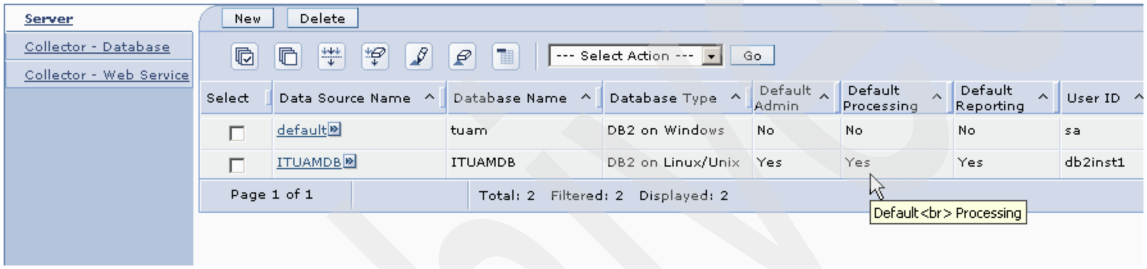


Figure 4-26 Tivoli Usage and Accounting Manager database settings

4.5.4 Initialize the Tivoli Usage and Accounting Manager database

Initializing the Tivoli Usage and Accounting Manager database creates and populates database tables and other database objects. Initializing the database is invoked from the ISC and the initialization is performed against the databases that are identified as the default administration data source using the **Set Admin** actions described in the previous section.

1. If you modify the data source definition or redefine the JDBC driver, you must restart the application server for ISC in order to pick up the new JDBC driver and database definitions.

```
/opt/ibm/tuam/ewas/bin/stopServer.sh server1
/opt/ibm/tuam/ewas/bin/startServer.sh server1
```

2. To initialize the Tivoli Usage and Accounting Manager database using the ISC menu select **Usage and Accounting Manager** → **System Maintenance** → **Database** → **Initialize Database**. The window shown in Figure 4-27 on page 96 is displayed.

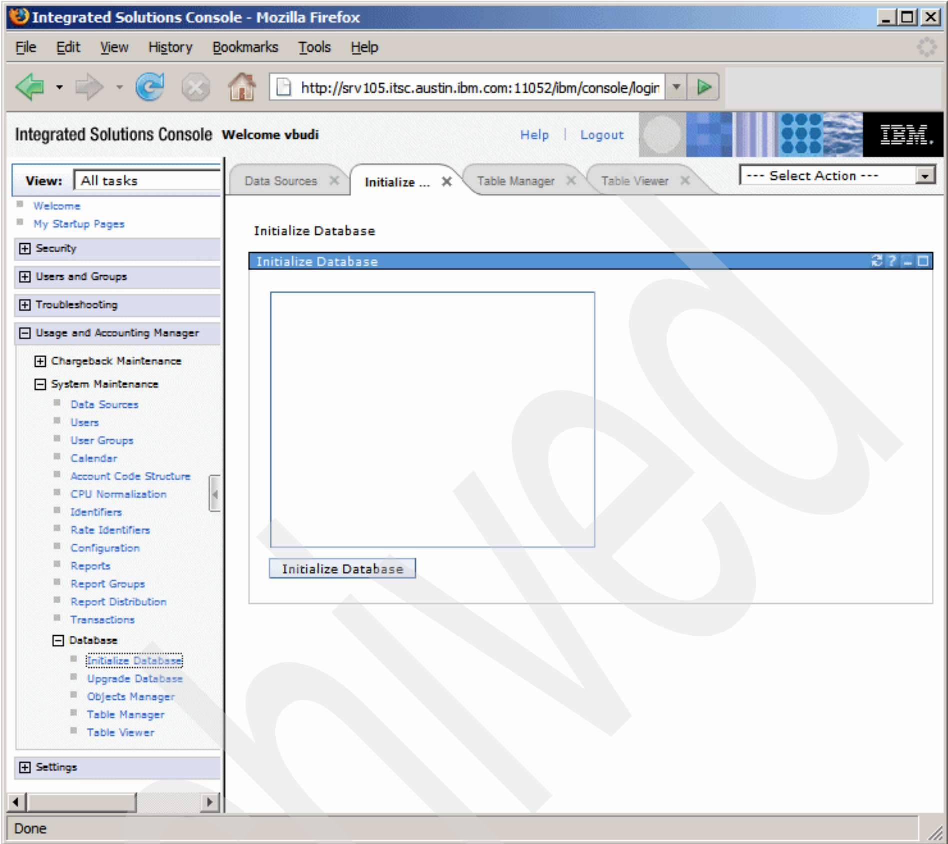


Figure 4-27 Initialize the Tivoli Usage and Accounting Manager database

3. Confirm the initialization of the database by clicking **Yes** (Figure 4-28).

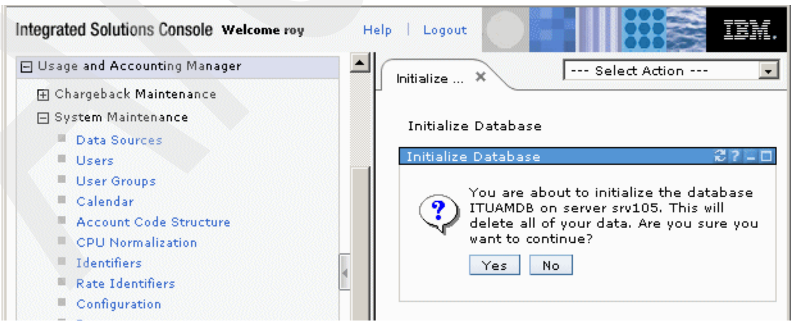


Figure 4-28 Confirmation window for database initialization

4. Review the results of initialization in the Initialize Database window (Figure 4-29).

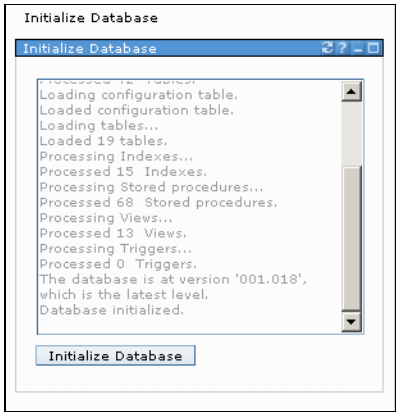


Figure 4-29 Results of Tivoli Usage and Accounting Manager database initialization

4.5.5 Run the samples for testing the installation

Before doing any further configurations in Tivoli Usage and Accounting Manager, run the provided samples:

```
/opt/ibm/tuam/bin/RunSamples.sh | tee RunSamples.log
```

We can ignore warning messages like the one in Figure 4-30 because SMTP is not configured in our environment.

```
WARNING: AUCCM5019E The process failed when sending e-mail through
ITUAM@ITUAMCustomerCompany.comSMTP from to
John.ITUAMUser@ITUAMCustomerCompany.com. Review the trace log to get
detailed information.
```

Figure 4-30 Ignoring the Warning from RunSamples.sh

To check for the results, from the ISC menu select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Log Files** as shown in Figure 4-31 on page 98.

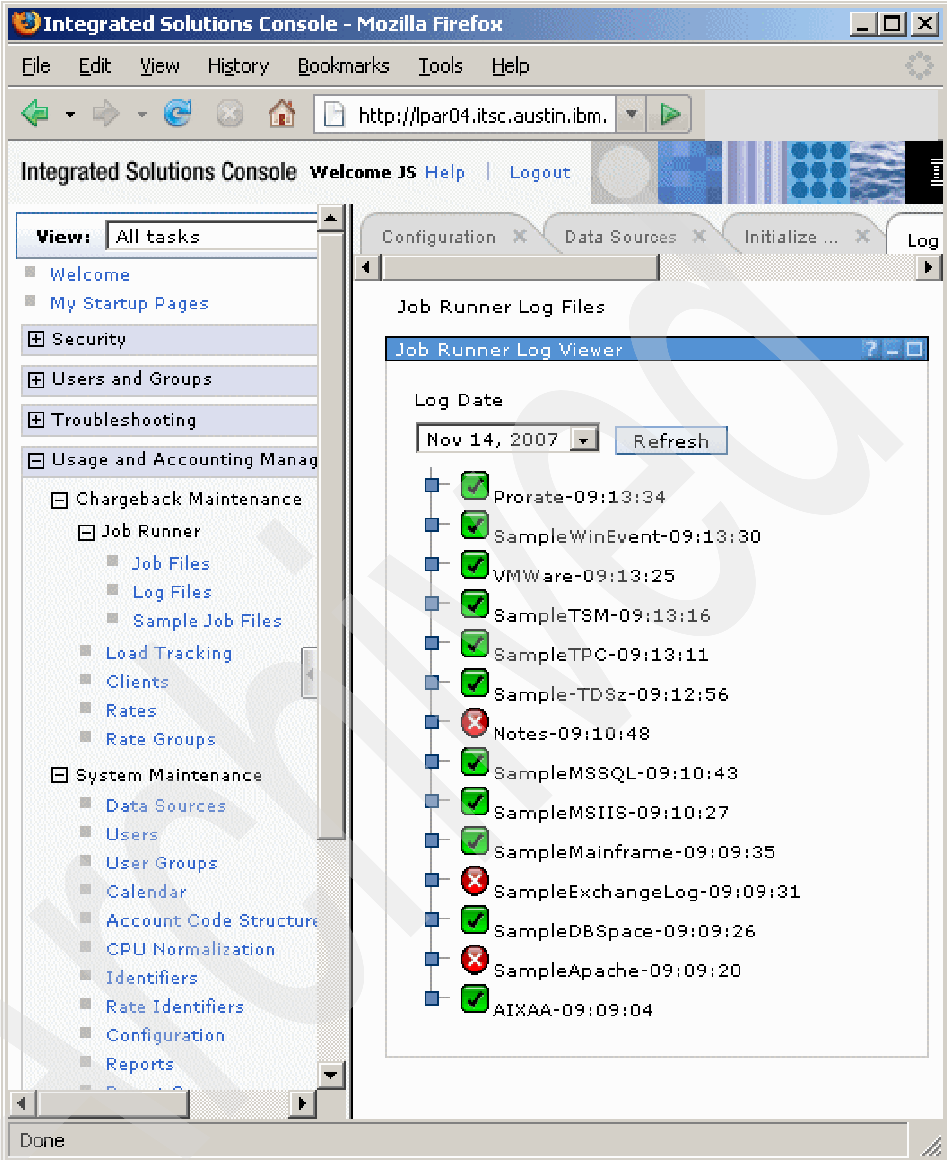


Figure 4-31 Checking the Log Files viewer after RunSamples.sh has finished

In our example the Notes job failed with this DB2 error message during loading:

```
AUCPE0202E The DBLoad process completed unsuccessfully with the
following exception: com.ibm.db2.jcc.c.lh: [ibm][db2][jcc][102][10040]
Non-atomic batch failure. The batch was submitted, but at least one
exception occurred on an individual member of the batch.
```

Other errors are caused by not having installed the enterprise collector package (ecp):

```
AUCIN0365E The following collector is not authorized to run in the
Usage and Accounting Manager Enterprise Edition: MS Exchange 2007
AUCIN0365E The following collector is not authorized to run in the
Usage and Accounting Manager Enterprise Edition: Apache
```

To do a final cleanup of the database, re initialize it as described in the previous section.

Alternatively, you can unload the data using the ISC menu selections **Usage and Accounting Manager** → **Chargeback Maintenance** → **Load Tracking** as shown in Figure 4-32. Deselect the check box for the **End Date** filter and mark all by clicking the **Check box** button.

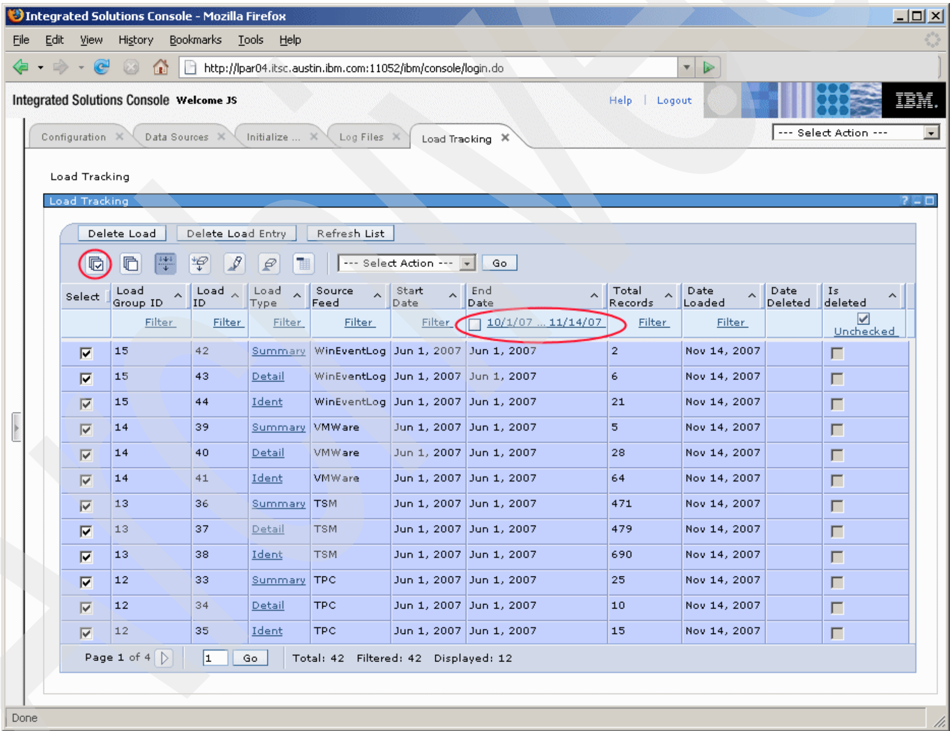


Figure 4-32 Unloading the RunSamples.sh data from the database

Press **Delete Load** and confirm the security question to delete the data.

4.5.6 Set up the processing directories

The processing directories define the path to find and store files for processing of usage and accounting data.

1. Configure the path by selecting the ISC options **Usage and Accounting Manager** → **System Maintenance** → **Configuration** → **Processing**. Update the path according to your installation. Figure 4-33 shows our path definitions.

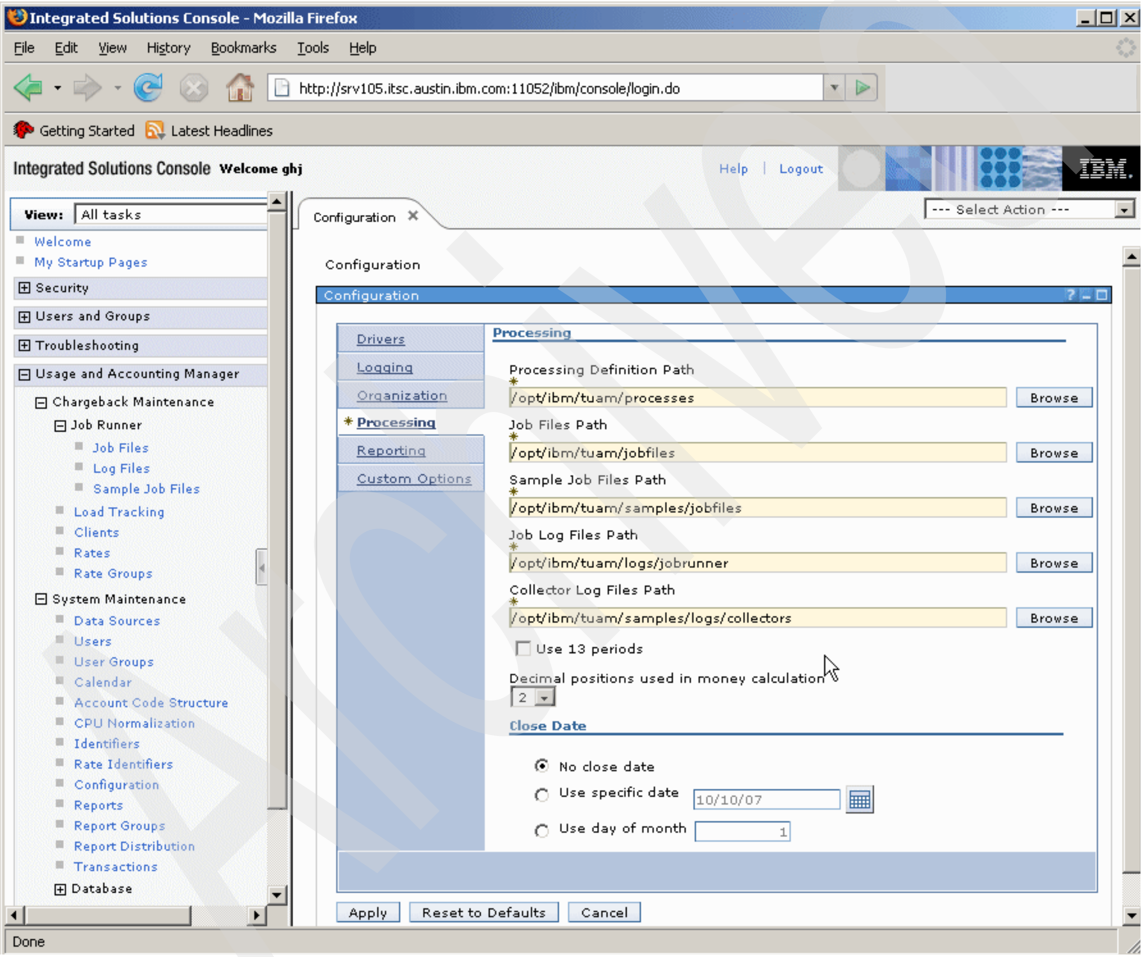


Figure 4-33 Configuring the paths used during processing

> **Note:** The original processing path from the sample directory is used by the runSamples program to generate sample data.

2. On the server, create the directories according to the path definitions that you have just configured. We did this from the command line session as shown in Figure 4-34. Note that the default directories for the Job File Path, Sample Job File Path, Job Log Files Path and Collector Log Files Path are already created when the Application server is installed.

```
# cd /opt/ibm/tuam
# mkdir processes
```

Figure 4-34 Create the processes directory

4.6 Verifying the database configuration

You can verify the Tivoli Usage and Accounting Manager database initialization from the ISC console.

4.6.1 Verify the tables created during initialization

Make the menu selections **Usage and Accounting Manager** → **System Maintenance** → **Database** → **Table Manager**. Figure 4-35 on page 102 shows the table list of the default administration database. There should be 42 tables in the list.

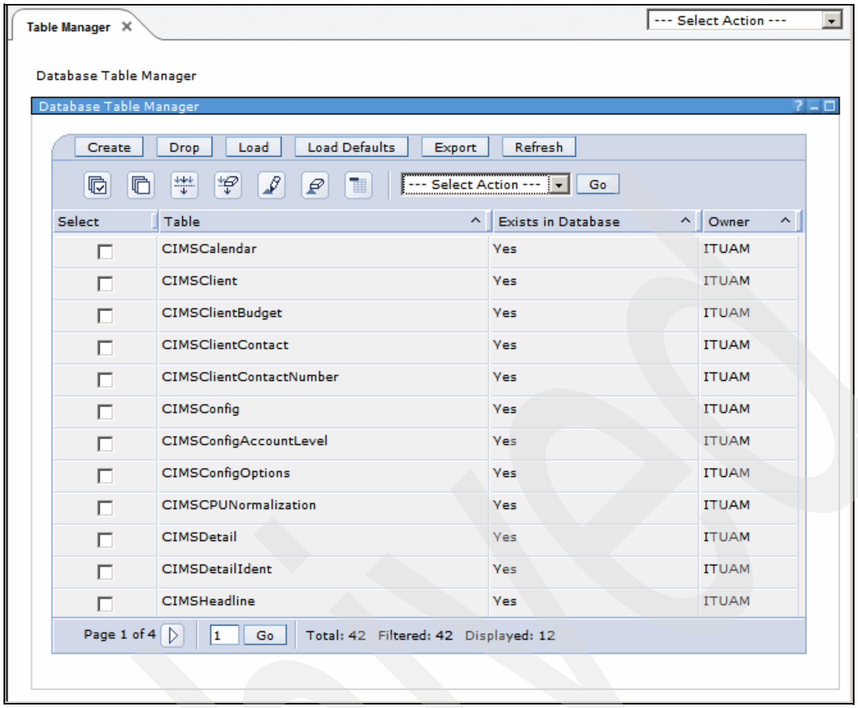


Figure 4-35 Table list

4.6.2 Verify the contents of the CIMSRate table

To verify the contents of the CIMSRate table, using the ISC menu, select **Usage and Accounting Manager** → **System Maintenance** → **Database** → **Table Viewer**. Expand the Database Tables tree and select the CIMSRate table from the Table Viewer main window. Click the **View Table** button to see the contents of the CIMSRate table (Figure 4-36 on page 103).

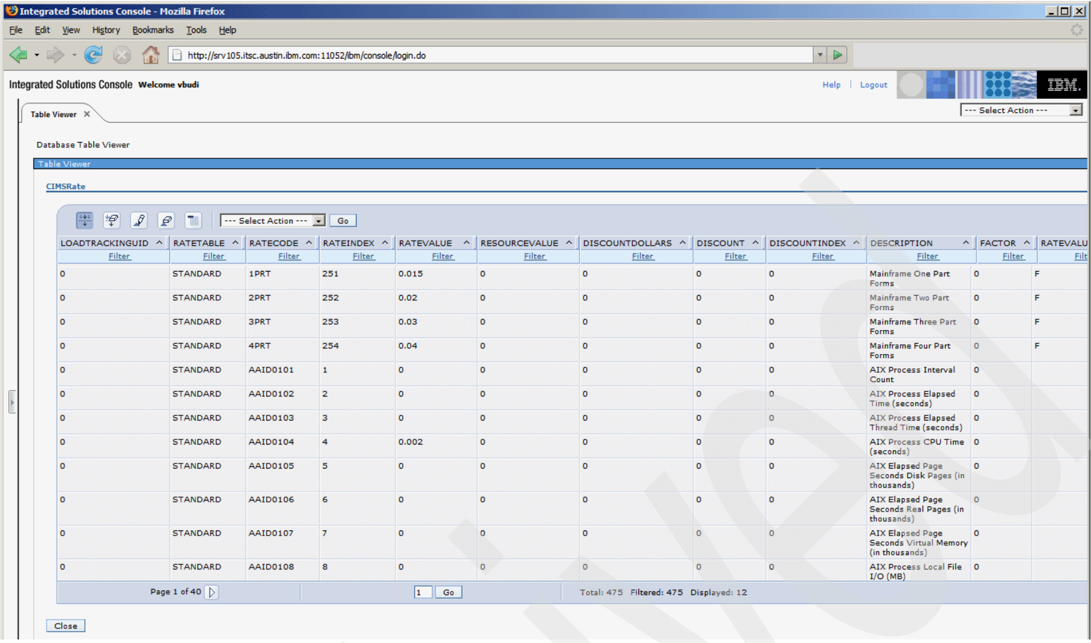


Figure 4-36 View the contents of the CIMSRate table

4.7 Installing and configuring the reporting server

We installed the reporting server on a separate Windows 2003 Server system (system SRV177 in our example). Prior to installing Tivoli Usage and Accounting Manager on the reporting server the following pre-requisites must be in place:

- Microsoft Internet Information Services (IIS) is required for the execution of the reporting application of Tivoli Usage and Accounting Manager.
- A current version of the Microsoft Installer package. We installed MSI30-KB884016. See:
  `http://support.microsoft.com/kb/884016`
- Microsoft .NET Framework Redistributable 2.0 is required for installing the Microsoft Report Viewer. See:
  `http://www.microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en`
- Microsoft Report Viewer Redistributable 2005 is required for the standard Usage and Accounting Manager reports. See:
  `http://www.microsoft.com/downloads/details.aspx?familyid=8a166cac-758d-45c8-b637-dd7726e61367&displaylang=en`

4.7.1 Install the Microsoft Installer

An up to date version of the Windows Installer software must be available on the Report server system.

1. Download the Windows Installer from:

   `http://www.microsoft.com/downloads/details.aspx?FamilyID=5fbc5470-b259-4733-a914-a956122e08e8&DisplayLang=en`

2. Execute the program WindowsInstaller-KB884016-v2-x86.exe to run the installation of the Windows Installer. The Welcome screen is displayed (Figure 4-37). Click **Next**.

Figure 4-37 Welcome screen for the Windows Installer installation

3. Accept the license and click **Next** (Figure 4-38 on page 105).

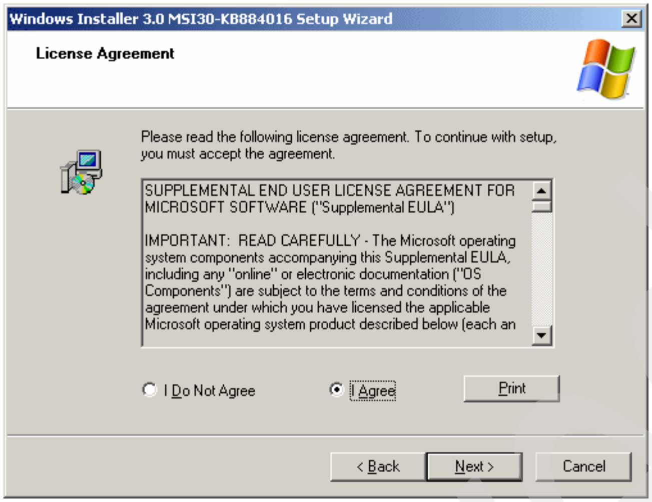


Figure 4-38 License agreement for the Windows Installer

4. Selected files on your system are backed up. The Windows Installer is installed and the completion window shown as in Figure 4-39 is displayed Click **Finish** to end the installation.

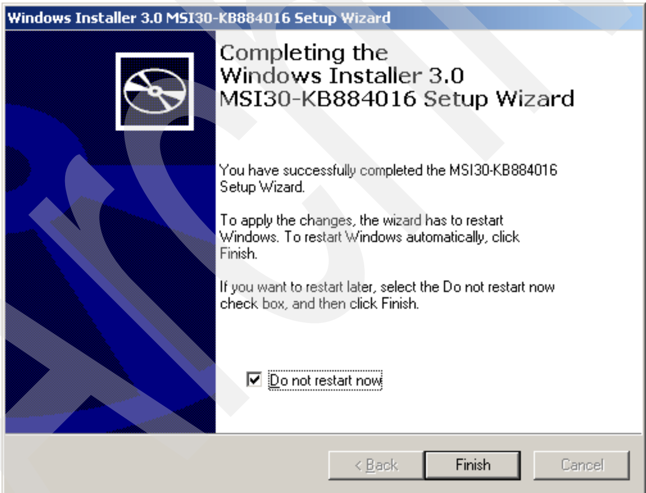


Figure 4-39 Completion of the installation for the Windows Installer software

4.7.2 Install Microsoft .NET Framework 2.0

The .NET Framework is required if you install Microsoft Report Viewer to view the standard Tivoli Usage and Accounting Manager reports in RDL format.

1. Download the installation package for the Report Viewer from:

   `http://www.microsoft.com/downloads/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5&DisplayLang=en`

2. Execute the downloaded program to start the installation and click **Next** at the Welcome screen. Accept the license agreement and click **Install** as in Figure 4-40. The installation progress window is displayed.

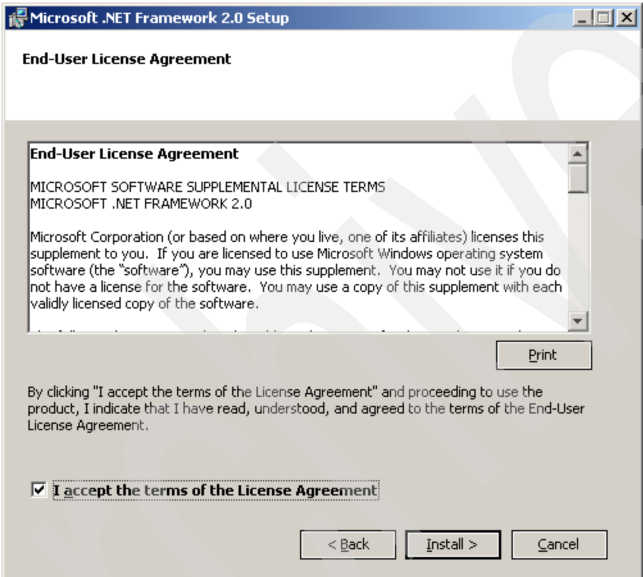


Figure 4-40 Accept the .NET license agreement and start the installation

3. The Setup Complete message is displayed when the installation is done (Figure 4-41 on page 107). Click **Finish** to end the installation.

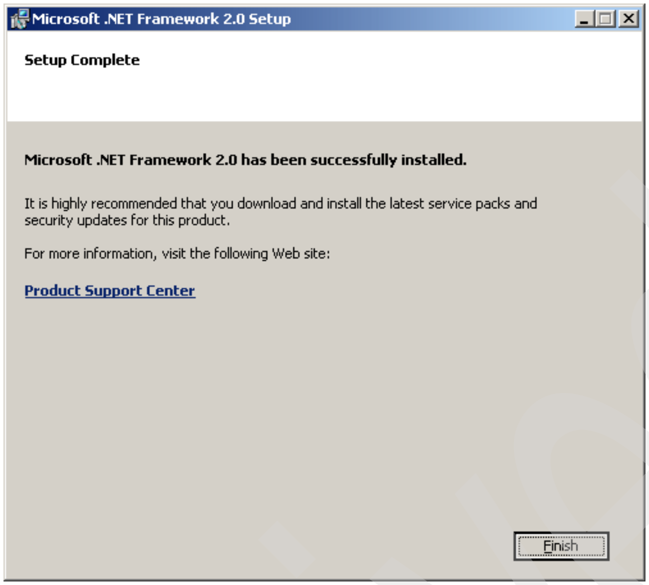


Figure 4-41 Setup complete for the .NET framework software

4.7.3 Install Microsoft Report Viewer 2005

The Microsoft Report Viewer is required for the standard Tivoli Usage and Accounting Manager reports (RDL format.)

1. Download the installation package for the Report Viewer from:

   `http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=8a166cac-758d-45c8-b637-dd7726e61367`

2. Save the downloaded file as ReportViewer.exe.
3. Execute the program ReportViewer.exe to install the Report Viewer. The Welcome screen is displayed (Figure 4-42 on page 108). Click **Next**.

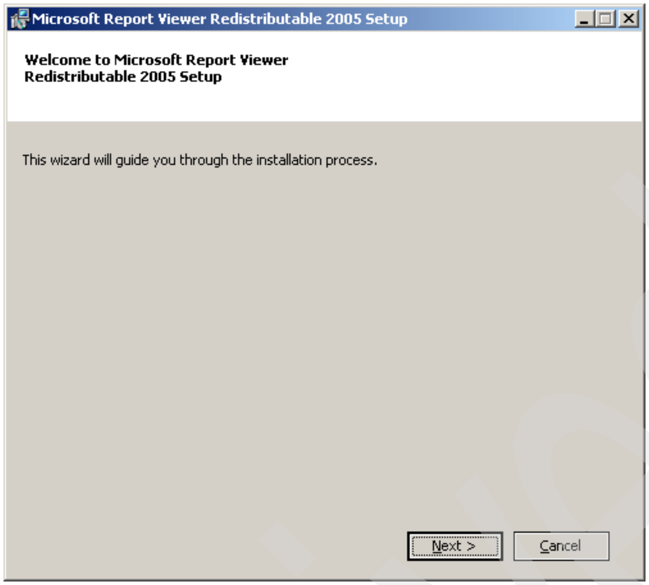


Figure 4-42 Welcome screen for the Report Viewer installation

4. Accept the license agreement and click **Install** (Figure 4-43).

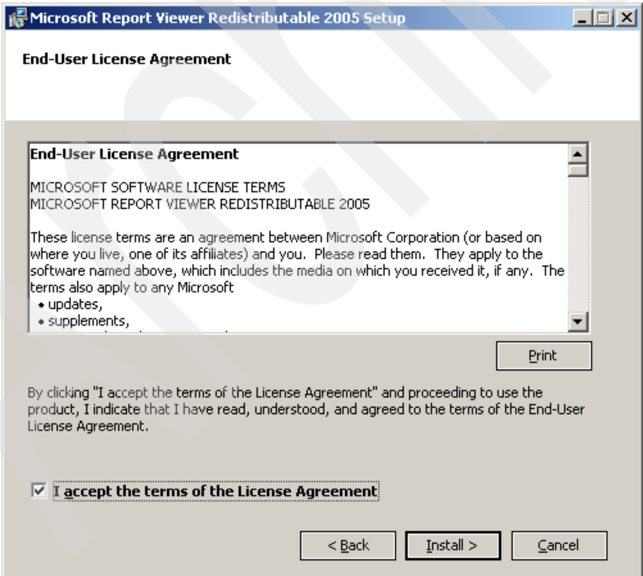


Figure 4-43 License agreement for the Report Viewer

5. Once successfully installed, the Setup Complete window is displayed as in Figure 4-44. Click **Finish** to end the installation.

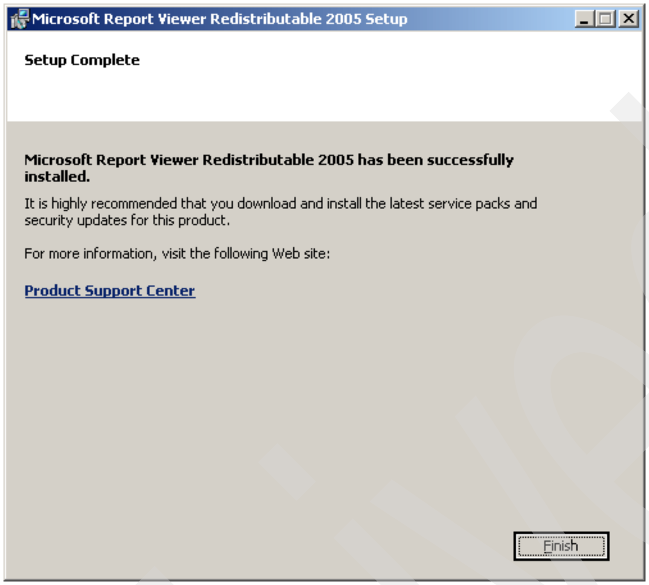


Figure 4-44 Successful installation of the Report Viewer

4.7.4 Install the Tivoli Usage and Accounting Manager report server

Install the reporting server using the Tivoli Usage and Accounting Manager enterprise edition Windows installation package. This installation package contains the reporting server as well as the Tivoli Usage and Accounting Manager Application server software, the ISC, embedded WebSphere Application Server, and the DB2 Universal Database V9.1 runtime client.

Make sure that you have the Microsoft Internet Information Server installed and active.

All the following files must exist in the same directory:

- EmbeddedExpress_wintel_ia32.zip
- ISCAE71_4_EWASv61.zip
- setup-tuam-ee-7-1-0-wintel_ia32.exe
- setup-tuam-wpc-7-1-0-windows_32_64.exe
- v9fp2_ALL_LANG_setup_32.exe

Install the report server by performing the following steps:

1. Execute program setup-tuam-ee-7-1-0-wintel_ia32.exe to install the Report server. Click **Next** at the Welcome screen. Accept the license agreement and click **Next** as in Figure 4-45.

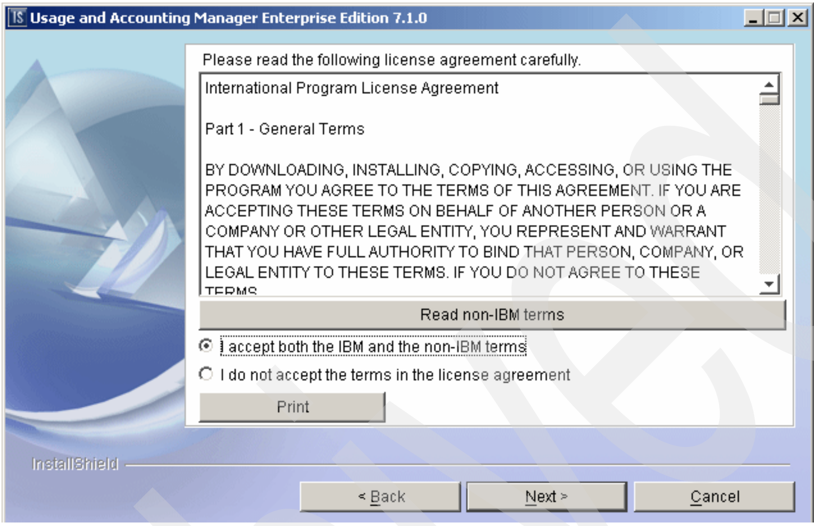


Figure 4-45 Accept license for the Tivoli Usage and Accounting Manager Report server

2. Install the Report server into the C:\IBM\tuam\ directory, which is the default directory (Figure 4-46). Click **Next**.

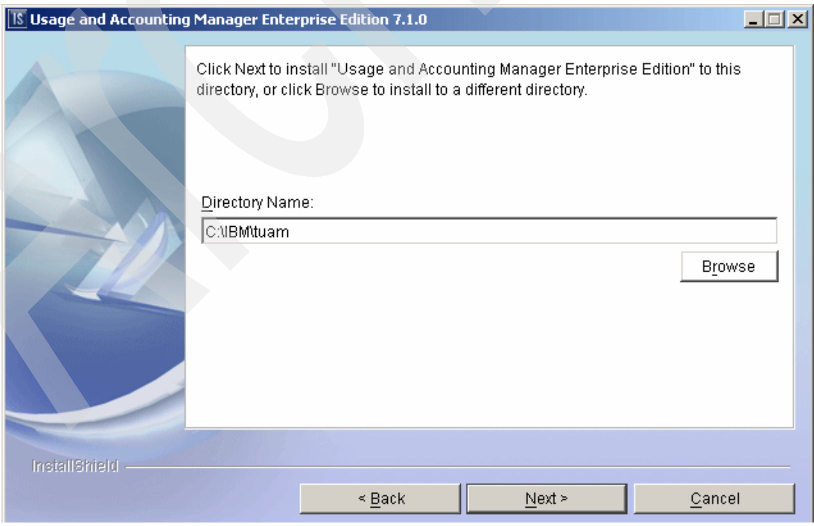


Figure 4-46 Specify the installation directory for the Report server software

3. Select the Windows Web Reporting option as shown in Figure 4-47. Click **Next**.

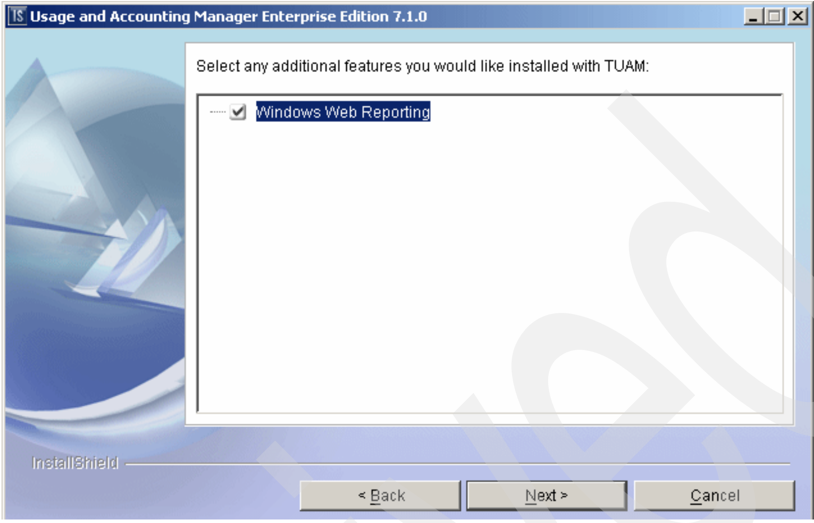


Figure 4-47 Click the windows Web Reporting option

4. Choose the virtual directory option as shown in Figure 4-48 and click **Next**.

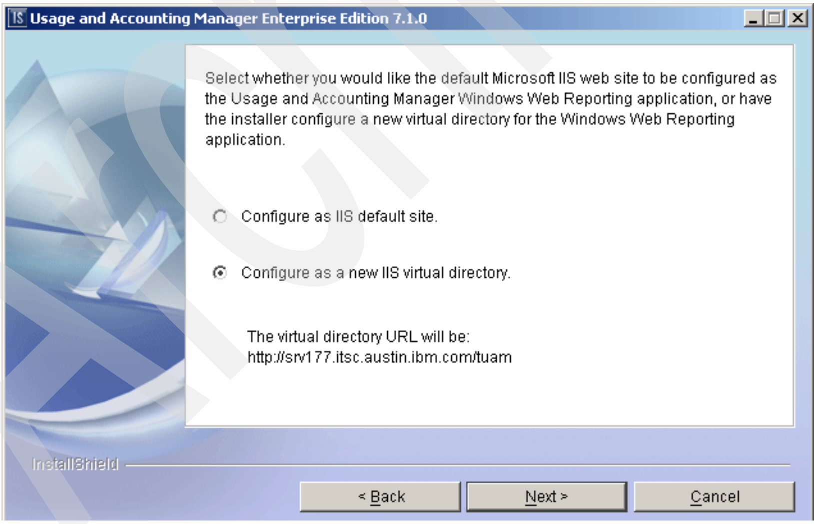


Figure 4-48 Select a new virtual directory for web reports

5. Click **Install** on the summary information screen. The installation progress indicator is displayed.

6. A task is automatically initiated to unpack the installed files as shown in Figure 4-49.

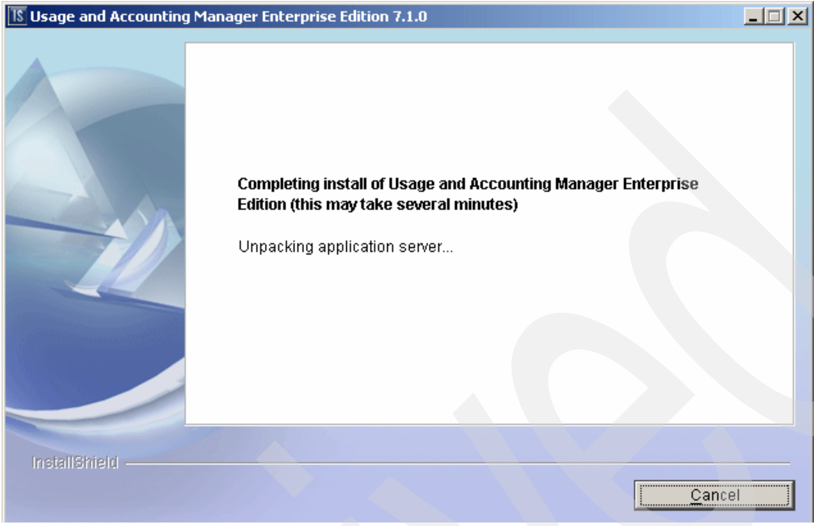


Figure 4-49 Unpacking of the Application server software on the Report server

7. Successful completion of the installation is indicated with the summary information shown in Figure 4-50. Click **Finish** to end the installation.

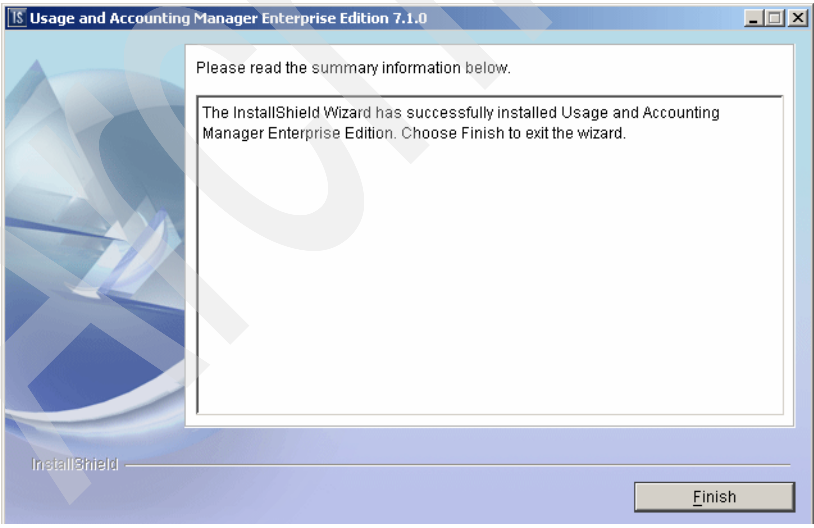


Figure 4-50 Successful installation of the Report server software

4.7.5 Configuring the Report server

Either of the following techniques can be used to configure the Report server:

- Configure the reporting server using the ISC, repeating some of the initial configuration steps performed for the administration server.
- Configure the reporting server using the configuration file from the administration server.

We consider it easier to use the file from the administration server, so we chose the second technique. Perform the following steps to configure your reporting server:

1. Back up the installed registry.tuam.xml file on the Windows reporting server. It is located in the C:\IBM\tuam\config directory. Use this backup if you need to restore the registry.tuam.xml file.
2. Manually copy the registry.tuam.xml file from the administration server system (Linux system SRV105 in our case) to the Windows Report server (SRV177). The file to copy is located in the /opt/ibm/tuam/config directory on the Linux server. Replace the registry.tuam.xml file on the Windows server.
3. Manually edit the registry.tuam.xml file in the C:\IBM\tuam\config directory on the Windows Report server and change the /IBM/TUAM/Settings.DynamicClassPath setting. This parameter should reference the JDBC driver that will be used to connect to the Tivoli Usage and Accounting Manager Database server. We used the DB2 Universal Database V9.1 JDBC drivers that are installed with the Report server, and changed the setting as in Figure 4-51. Save the updated file.

```
<entry
key="/IBM/TUAM/Settings.DynamicClassPath">;C:\IBM\tuam\DB2RTC\java\d
b2jcc.jar;C:\IBM\tuam\DB2RTC\java\db2jcc_license_cu.jar</entry>
```

Figure 4-51 Setting the DynamicClassPath to reference the DB2 UDB JDBC drivers

4.8 Verifying the installation of the Report server

Verify the installation of the Report server using the following techniques:

- 4.8.1, “Verify the files created during installation” on page 114
- 4.8.2, “Verify the application status in IIS” on page 114
- 4.8.3, “Connect to reporting Web application” on page 115

4.8.1 Verify the files created during installation

Open Windows Explorer and expand the C:\IBM\tuam\bin directory to verify that files have been installed. Our results are shown in Figure 4-52.

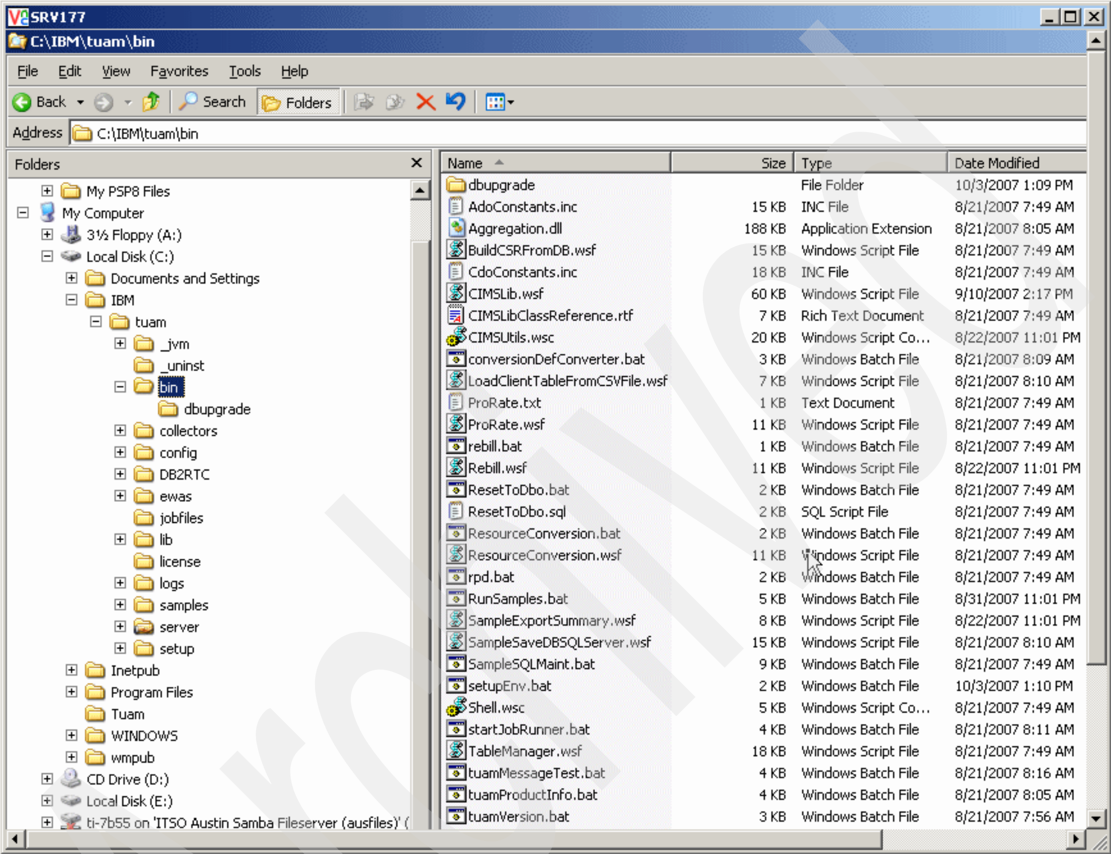


Figure 4-52 Verify installation of files to the bin directory on the Windows Report server

4.8.2 Verify the application status in IIS

On the Report server, open the Microsoft Internet Information Services (IIS) Manager using **Start → All Programs → Administrative Tools → Internet Microsoft Internet Information Services (IIS) Manager**. Expand the trees to display the Application Pools and the Web Sites as shown in Figure 4-53 on page 115. Confirm the existence of the Tivoli Usage and Accounting Manager entries.

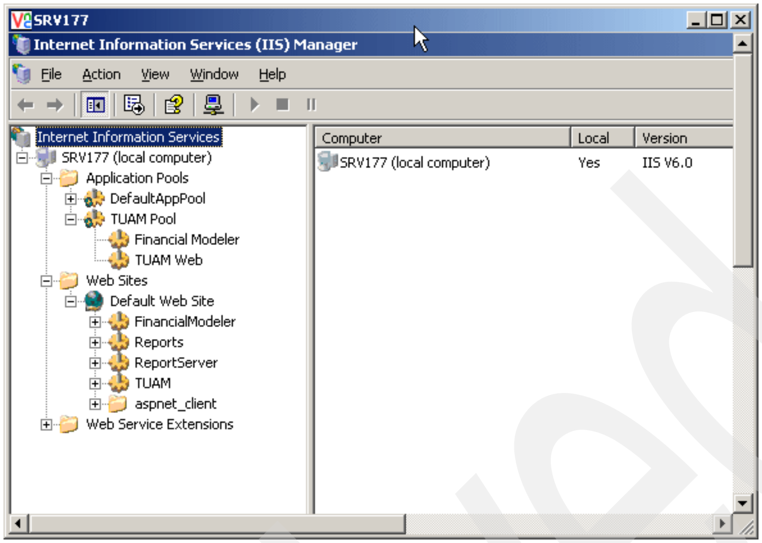


Figure 4-53 Verify the installation of the Tivoli Usage and Accounting Manager Web components

4.8.3 Connect to reporting Web application

Open a browser, and point it at the Tivoli Usage and Accounting Manager Web application URL. In our case this is:

```
http://srv177/tuam/
```

The browser window displays the initial Web reporting option screen as in figure Figure 4-54 on page 116.

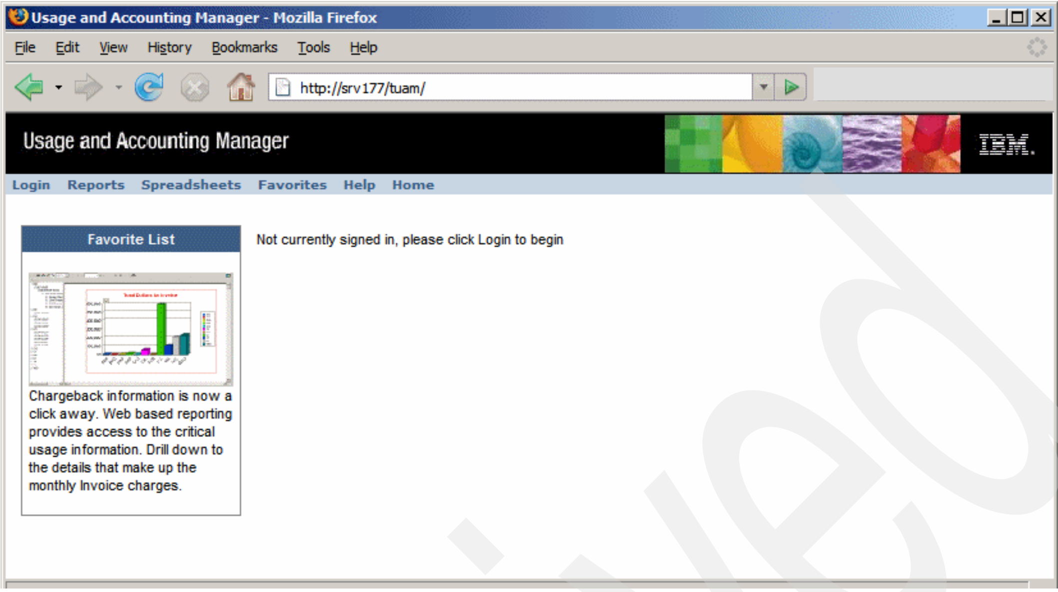


Figure 4-54 Initial display for the Tivoli Usage and Accounting Manager report application

Select **Login** and enter credentials. The default userid is `admin` with the password of `password` (Figure 4-55).

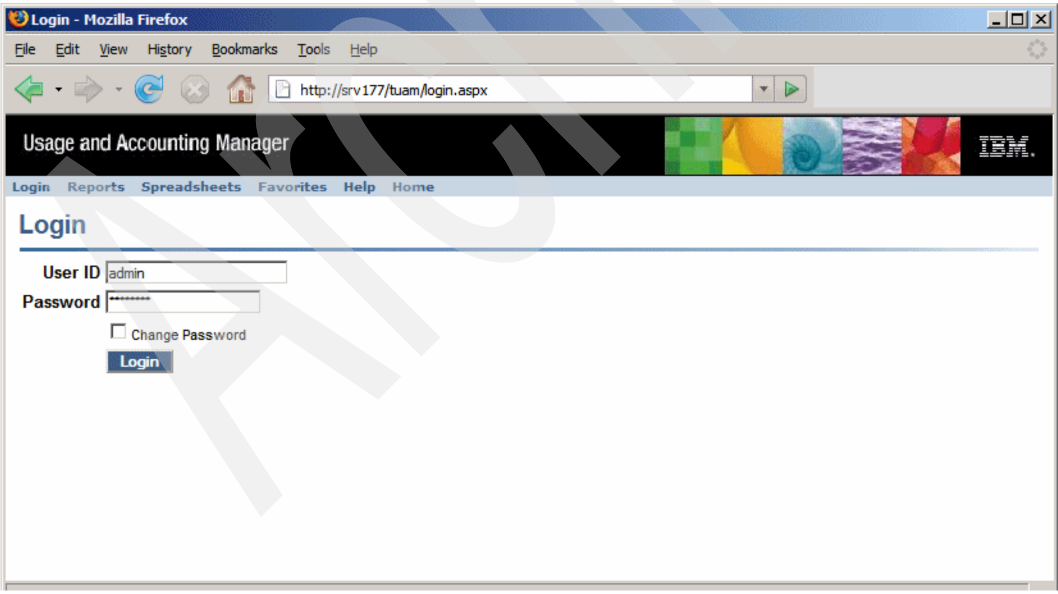


Figure 4-55 Logging in

Select **Reports** → **Run Reports** to see a list of report groups as in Figure 4-56.

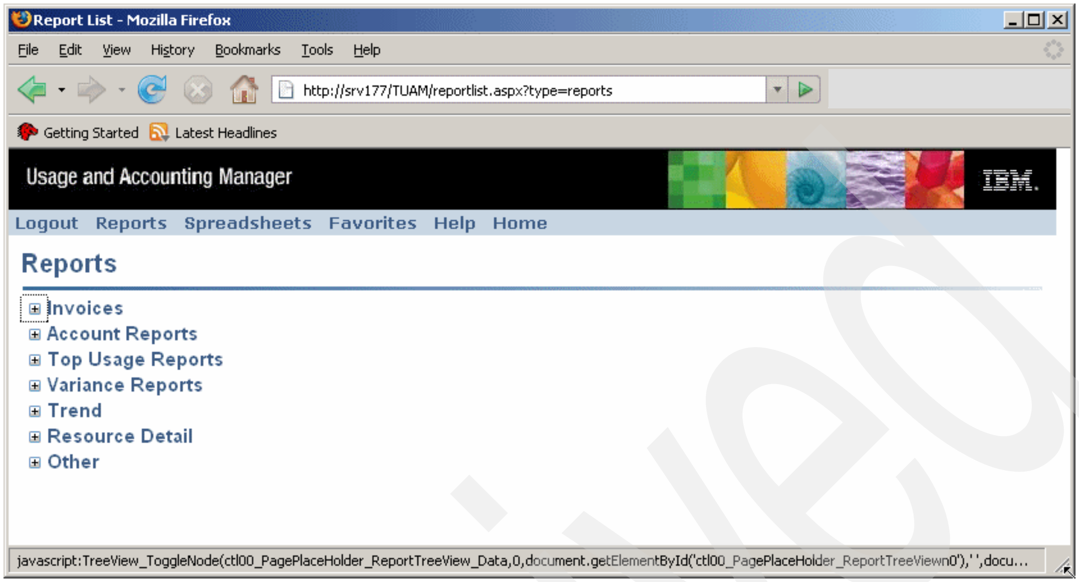


Figure 4-56 List of report groups

4.9 BIRT reporting installation and verification

We tried the sample BIRT reports in our Eclipse environment. To follow along with our example perform these steps:

1. Create a report project called ITUAM as shown in Figure 4-57.

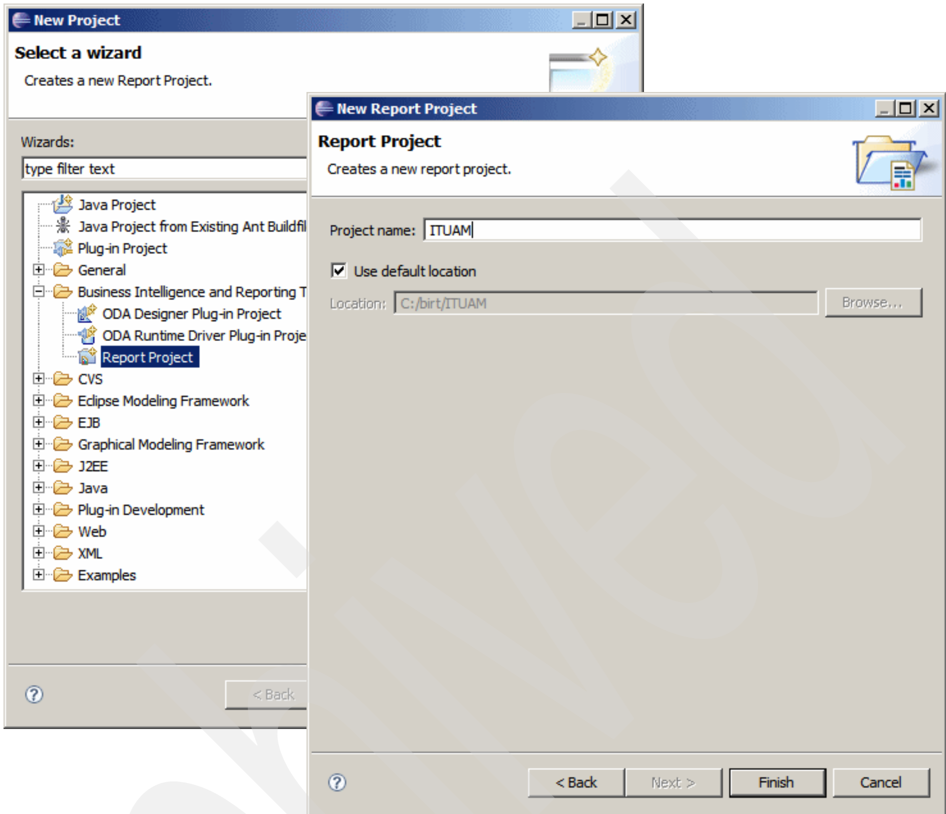


Figure 4-57 New project for reporting

2. In the report project, import the reporting from the file system as shown in Figure 4-58 on page 119. Import all files into the same directory from the following paths:

   – \IBM\TUAM\server\reportsbirt\db2\standard
   – \IBM\TUAM\server\reportsbirt\resources

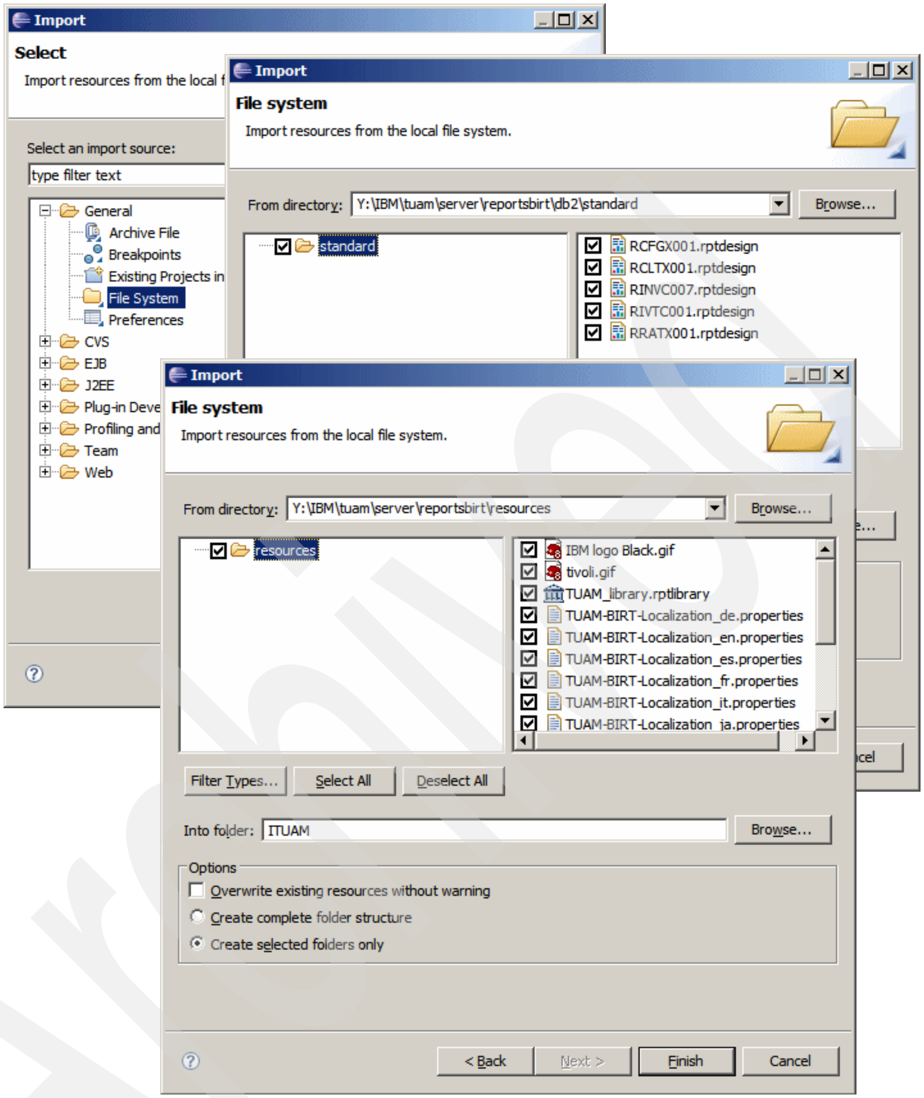


Figure 4-58 Project import from file system

3. Figure 4-59 on page 120 shows the project report contents once all the files are imported. There are currently 5 pre-defined reports, as indicated by the file extension `rptdesign`.

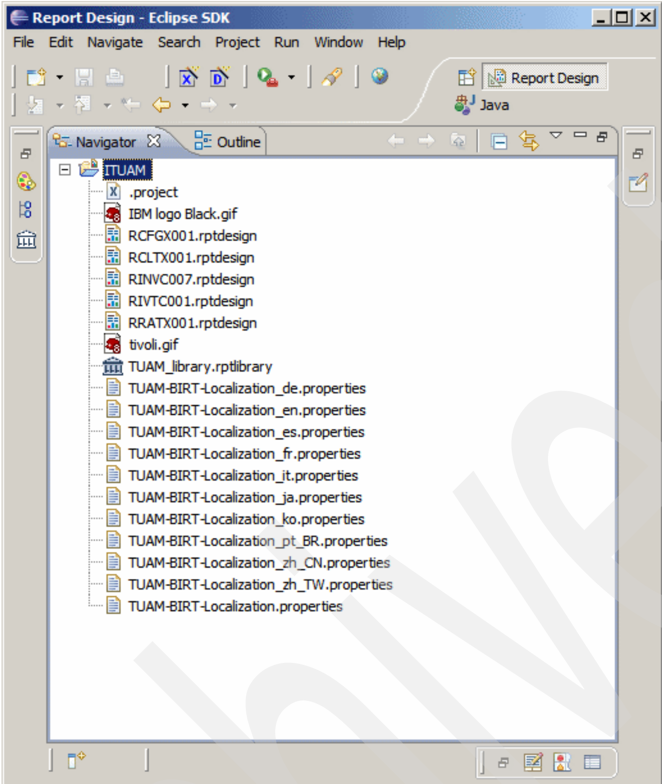


Figure 4-59 Project report contents

4. There are several steps required to run the report:
   a. Modify the TUAM_library.rptlibrary, which contains the definition of the data access and data sources (Figure 4-60 on page 121).

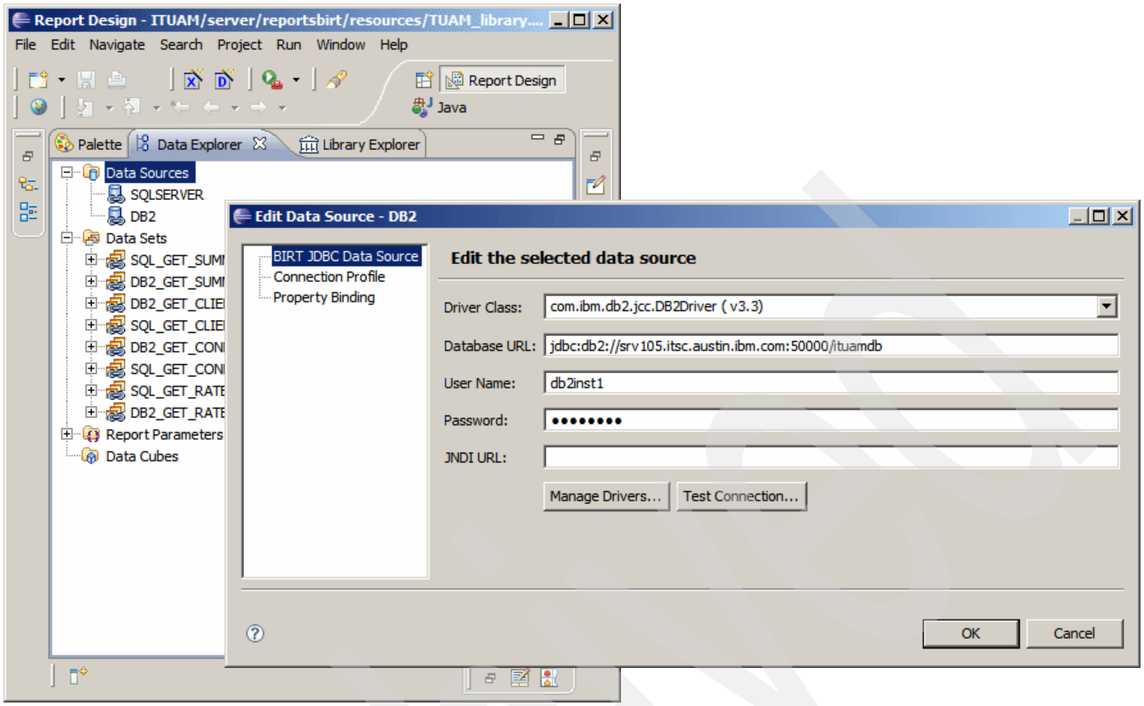


Figure 4-60 TUAM_library definition

b. The data sets are defined with the qualifier of DB2ADMIN. If you define the Tivoli Usage and Accounting Manager database with another name, each dataset must be modified (Figure 4-61 on page 122).

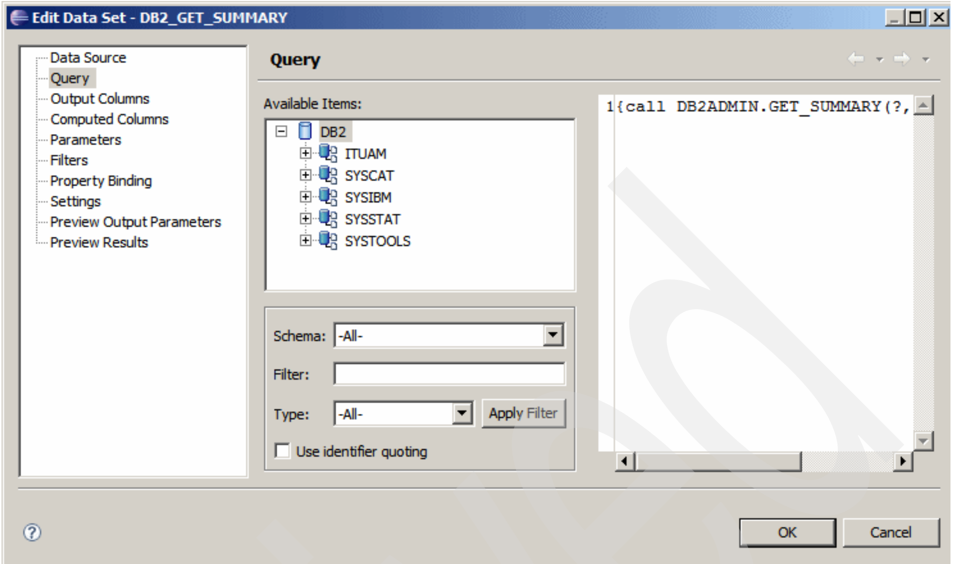


Figure 4-61 Modifying the report query

Sample reports

The following five figures show examples of some of the reports from the BIRT reporting system:

- Configuration summary report (Figure 4-62 on page 123)
- Client report (Figure 4-63 on page 123)
- Invoice report (Figure 4-64 on page 124)
- Invoice total report (Figure 4-65 on page 125)
- Rate report (Figure 4-66 on page 126)

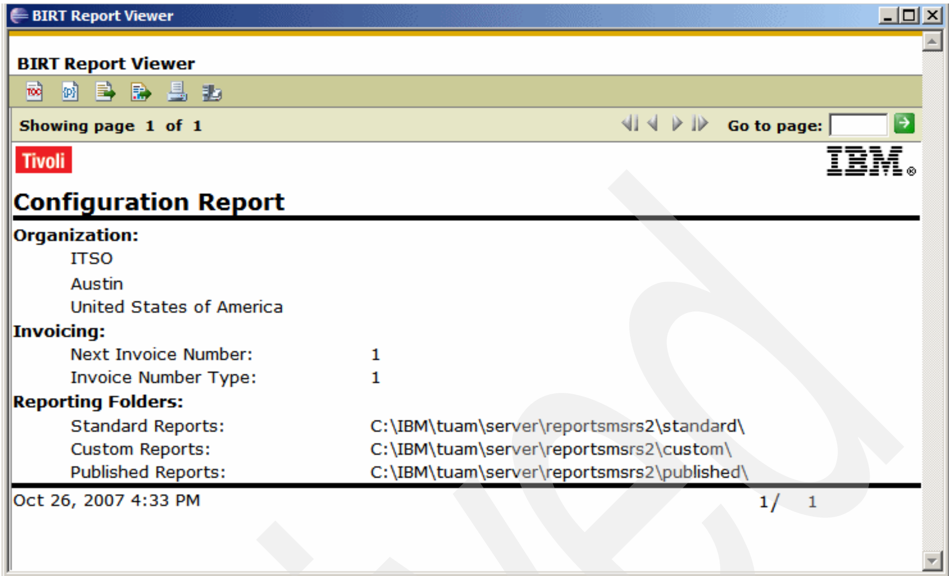


Figure 4-62 Configuration report

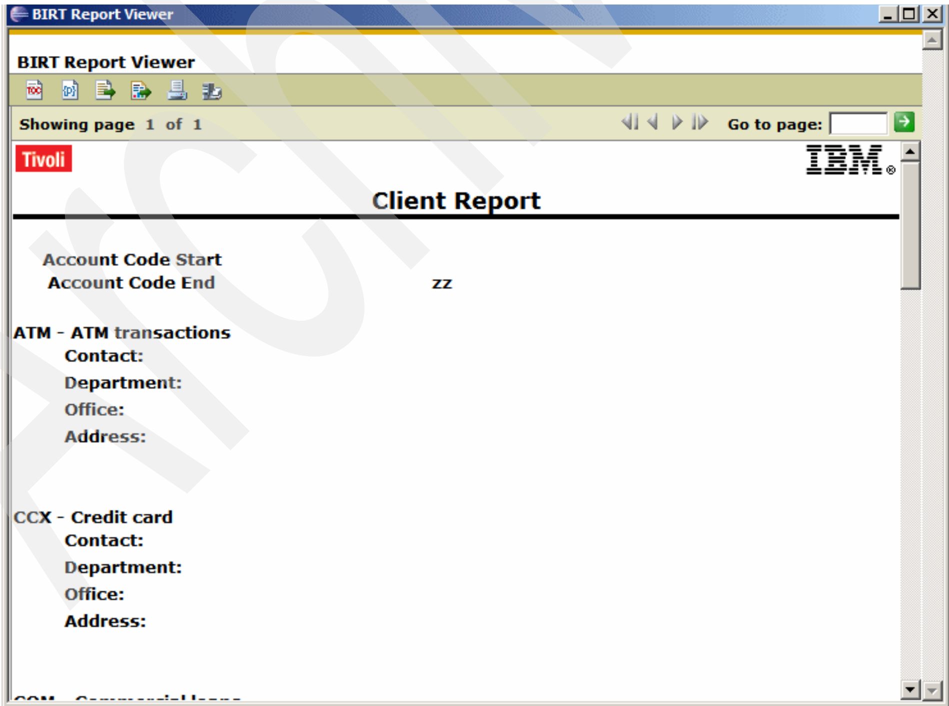


Figure 4-63 Client report

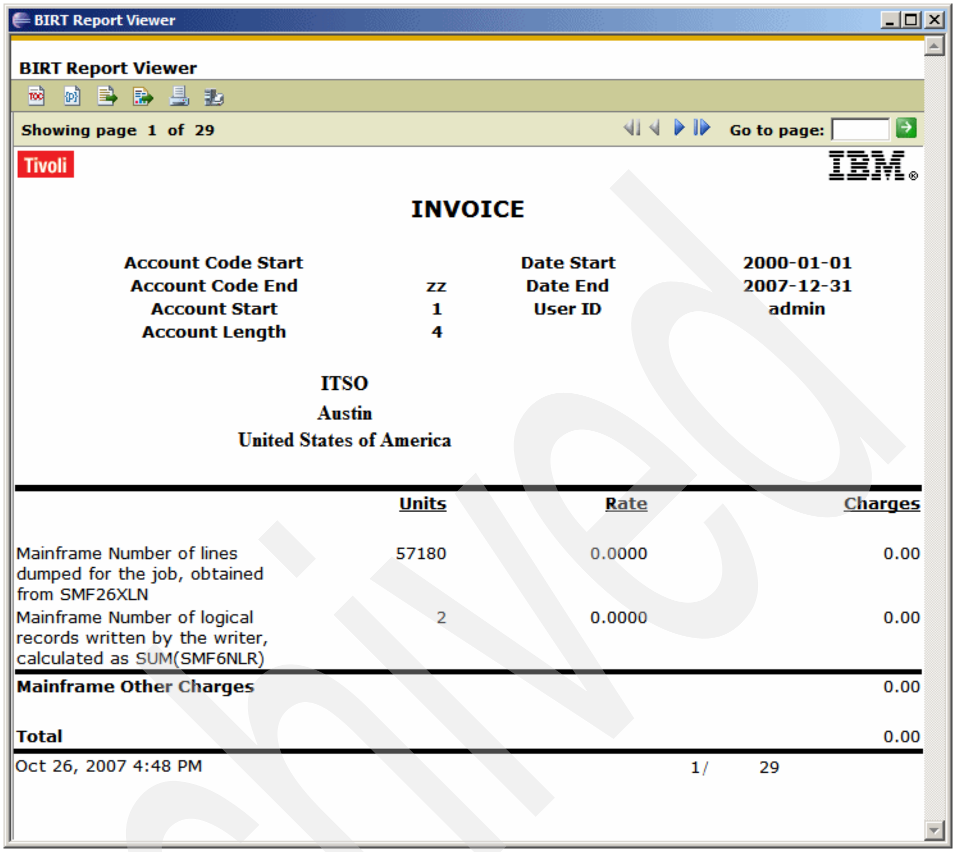


Figure 4-64 Invoice report

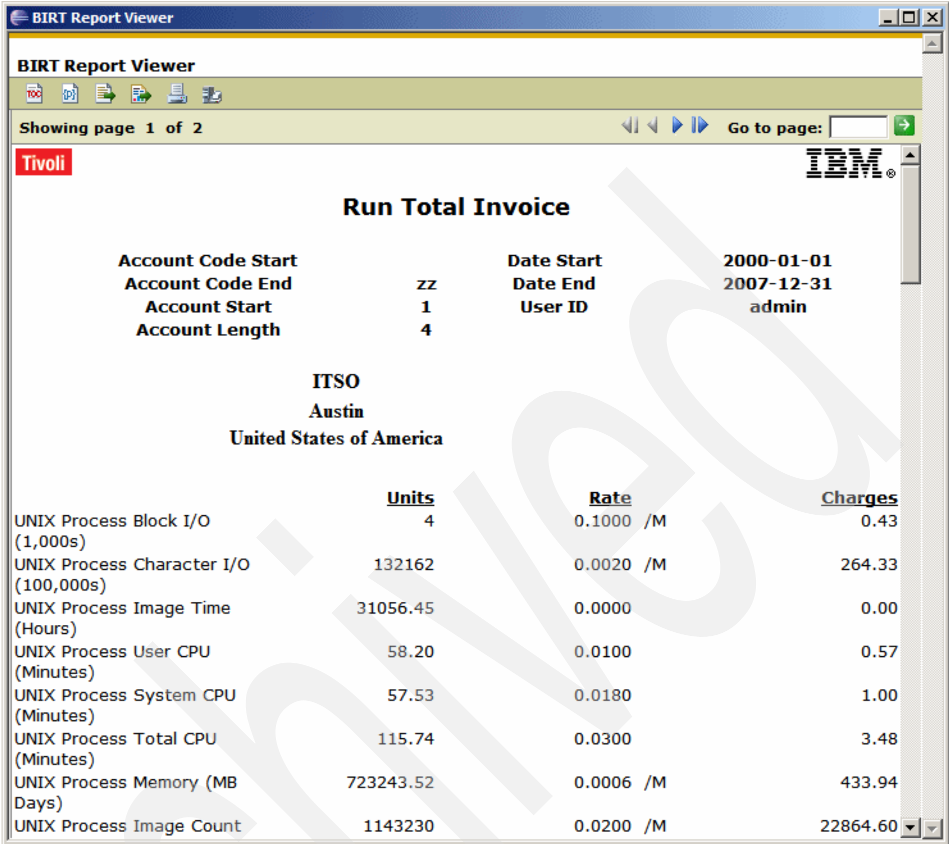


Figure 4-65 Invoice total report

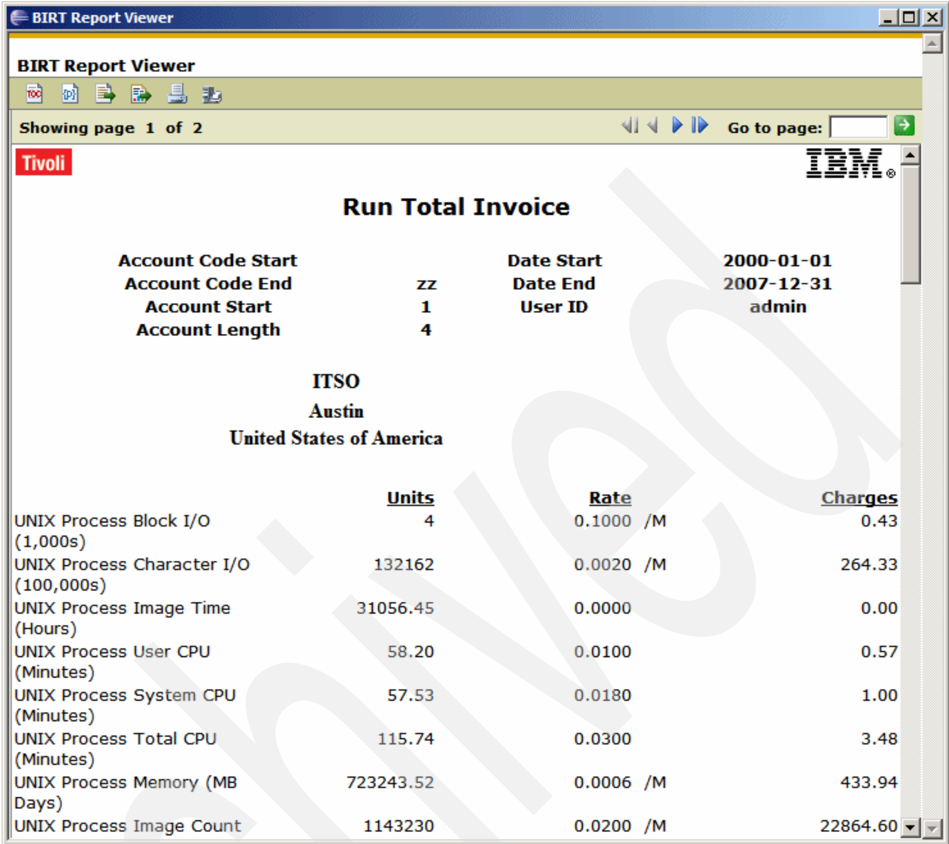


Figure 4-66 Rate table

5

Operating system data collection scenario

This chapter describes a basic data collection scenario. Files that are created by the AIX data collector and the Windows process data collector are transferred to the administration server and processed. This chapter contains the following sections:

5.1 Architecture for the basic data collection scenario

This section describes the hardware and software architecture utilized for the basic data collection scenario. We continue to use the Tivoli Usage and Accounting Manager infrastructure described in Chapter 4, “Installation and configuration” on page 73, with the addition of data collection on an AIX server and on a Windows desktop system. Thus we have an administration and processing server installed on a Linux system (srv105), the reporting server installed on a Windows 2003 system (srv177) and AIX and Windows systems running the Tivoli Usage and Accounting Manager data collector (lpar04 and 3c-000-c respectively). The scenario is shown in Figure 5-1.

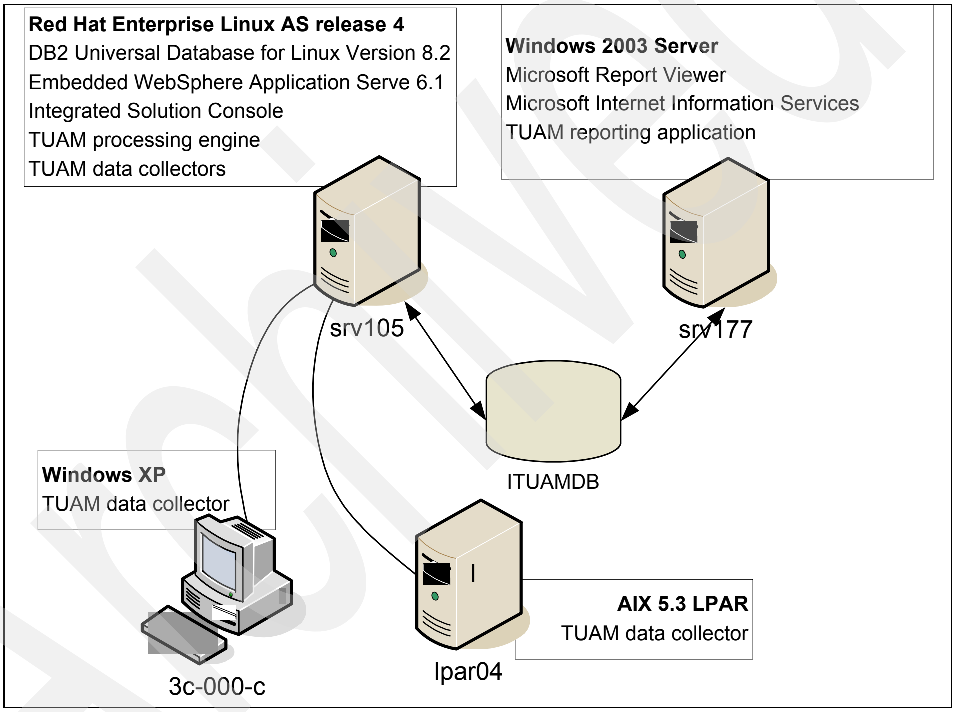


Figure 5-1 Basic data collection scenario

5.2 AIX data collection

This section describes operation of the Tivoli Usage and Accounting Manager data collection and processing from an AIX server. The topics covered are:

5.2.1 Remote installation of the AIX data collector

Remote installation helps you to quickly deploy data collectors to multiple systems. Tivoli Usage and Accounting Manager supplies a sample job to perform remote deployment. The job is executed under the control of the Tivoli Usage and Accounting Manager Job Runner. The customized job that we used for deployment to AIX is supplied in the appendix, under “Sample job for remote deployment to AIX” on page 286.

1. The Tivoli Usage and Accounting Manager remote installation process uses secure shell (ssh) and key authentication to perform the deployment. Verify that ssh-keys have been configured correctly between the processing server and the systems that you want to deploy the data collector to. From the processing server, run the **ssh** command to connect to the target system. The results of the **ssh** command in Figure 5-2 indicate that **ssh** is operating correctly.

```
[root@srv105 .ssh]# ssh lpar04 "hostname;uptime"
The authenticity of host 'lpar04 (9.3.5.114)' can't be established.
RSA key fingerprint is 33:13:31:2a:73:a1:54:d7:39:f6:c4:c7:54:64:34:7b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'lpar04,9.3.5.114' (RSA) to the list of known
hosts.
lpar04
  10:46AM   up   1:53,  2 users,  load average: 1.02, 1.03, 1.04
```

Figure 5-2 Results of the ssh command

If the ssh-key is not set up, you will be asked for a password. We use the key to perform data transfer later. You can set up the ssh keys on a UNIX system as follows:

a. From the target system, such as our lpar04 AIX system, run the **sftp** command to the Tivoli Usage and Accounting Manager server. Copy the Tivoli Usage and Accounting Manager server rsa key from $HOME/.ssh/id_rsa.pub

b. Confirm the RSA key fingerprint.

c. Add the key from the file to an authorization file.

The sample procedure is shown in Figure 5-3.

```
# cd /tmp
# sftp srv105
Connecting to srv105...
The authenticity of host 'srv105 (9.3.5.105)' can't be established.
RSA key fingerprint is
34:0b:18:19:d1:a4:a1:04:e9:59:8a:ef:af:61:ba:07.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'srv105,9.3.5.105' (RSA) to the list of
known hosts.
root@srv105's password:XXXXXXXX
sftp> get .ssh/id_rsa.pub
   Fetching /root/.ssh/id_rsa.pub to id_rsa.pub
   /root/.ssh/id_rsa.pub                          100%  241
   0.2KB/s   00:00
sftp> quit
# cd $HOME/.ssh
# cat /tmp/id_rsa.pub >> authorized_keys
```

Figure 5-3 Establish ssh-key on the Tivoli Usage and Accounting Manager client

2. To create the remote installation job file use the ISC menu. Select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files** and click **New**. Enter the name of the new deployment job that you are creating as in Figure 5-4 on page 131 and click **OK**. After the job file is created you receive an information message stating `Successfully copied file`. This indicates that the job file is created and contains a skeleton job file template.

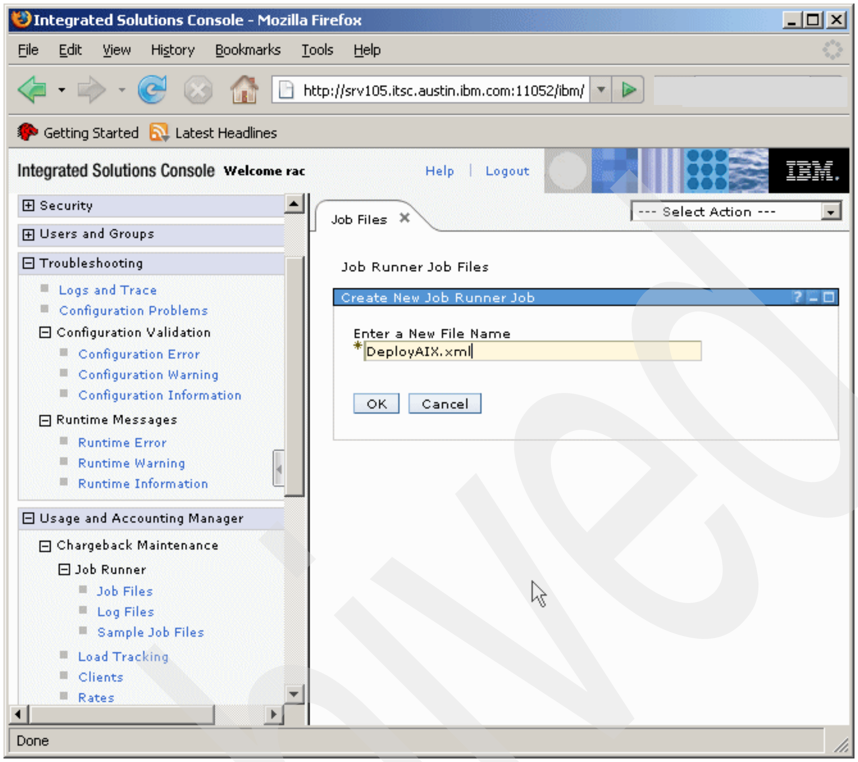


Figure 5-4 Create the AIX deployment job file

3. Tivoli Usage and Accounting Manager provides a sample deployment job file for AIX (and other flavors of UNIX). We use the contents of this sample to overwrite the contents of our new AIX deployment job file. From the ISC menu select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Sample Job Files**. Scroll through the list of sample job files and select SampleDeployUnixLinuxCollector.xml. The contents of the SampleDeployUnixLinuxCollector.xml are displayed in the main window.
4. Right-click in the main window to display the pop-up menu; select the menu options **Select All** and **Copy** as shown in Figure 5-5 on page 132.

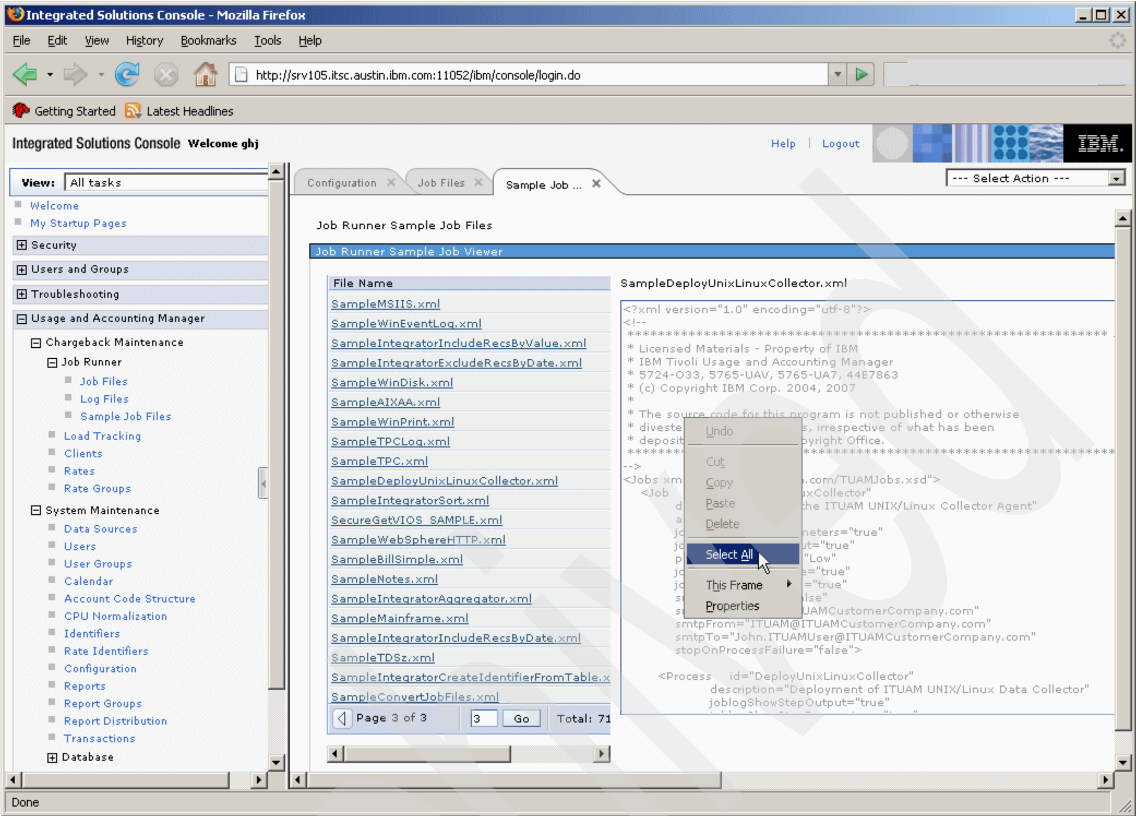


Figure 5-5 Select and copy text from the sample deployment job file

5. Return to the Job Files tab where you are editing the new AIX deployment script. Right-click in the main window to display the pop-up menu; select the menu options **Select All** and **Paste** as shown in Figure 5-6 on page 133. The contents of the sample deployment job file have been copied into the new AIX deployment job file.

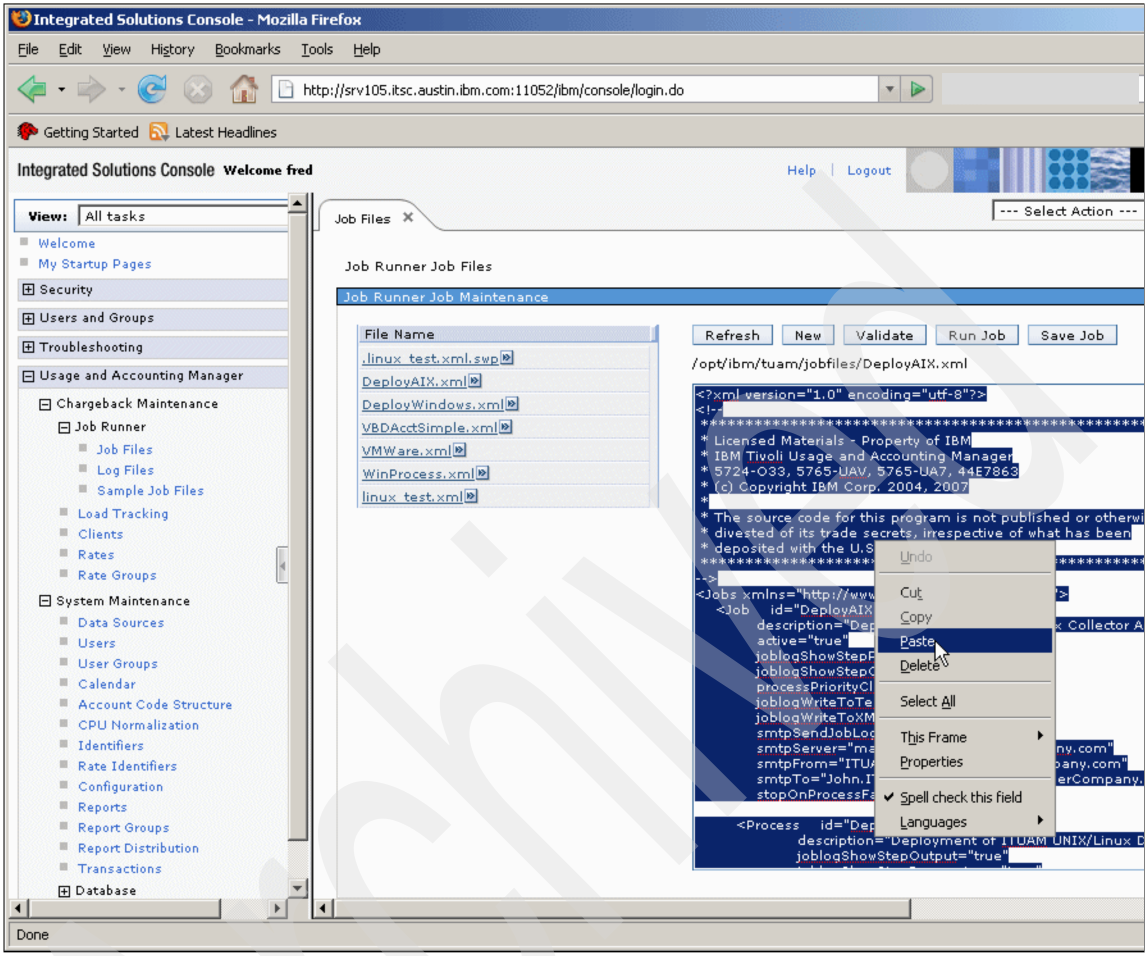


Figure 5-6 Paste the contents of the sample deployment job file into the new AIX deployment job file

6. Change the AIX deployment job file parameters, updating the values for Host, UserId, Password, KeyFilename, Manifest, RDPParameters, and SourcePath. Make the values appropriate for the system you are deploying to. Example 5-1 identifies the parameters we customized and the values we used.

Example 5-1 Customize the deployment parameters

```
<Job    id="DeployAIX"
<!-- SUPPLY hostname OF TARGET PLATFORM/-->
    <Parameter Host           = "lpar04"/>
<!-- userid must be set to root/-->
    <Parameter UserId         = "root"/>
```

```
<!-- SUPPLY root PASSWORD ON TARGET PLATFORM/-->
    <Parameter Password       = "password"/>
    <Parameter KeyFilename    = "/root/.ssh/id_rsa"/>
<!-- DEFINE Manifest TO MANIFEST XML FOR TARGET PLATFORM/-->
    <Parameter Manifest       = "DeploymentManifest_aix5.xml"/>
<!-- DEFINE INSTALLATION PARAMETERS,
       path:   must be defined to the directory path where UNIX/Linux Collector
               will be installed on target platform /-->
     <Parameter RPDParameters  =
"path=/opt/ibm/tuam/collectors/Unix;user=root;"/>
     <Parameter SourcePath     = "%HomePath%/collectors/unixlinux"/>
```

Note:

- Some parameters are shipped with comment tags. These comment tags should be removed if you require the parameter value to be utilized for the deployment. Commented text in the job file starts with the `<!--` characters and ends with the `-->` characters.
- The SourcePath value provided in the sample deployment job is not consistent with the default installation path. Customize it to match your installation.

7. Before you run the customized job file, check the syntax of the XML by using the **Validate** button. The `Validated successfully` information message will be displayed if there are no syntax errors. Correct any errors if the job file does not validate successfully.
8. Save the job file using the **Save Job** button.
9. Run the job file using the **Run Job** button. Job failure is discussed in 9.4, “Job runner debugging” on page 282.
10. The message `The job completed successfully` is displayed when the job runs without errors.

5.2.2 Verifying the AIX data collector deployment

Two techniques are used to verify the deployment, namely a directory listing, and the crontabs file listing.

1. Log on to the system that the data collector was deployed to.
2. List the contents of the directory that contains the data collector software. This is the directory specified by the RPDParameters value in the deployment job file. The command and results are shown in Figure 5-7 on page 135.

```
# ls -al /opt/ibm/tuam/collectors/Unix
total 96
drwxr-xr-x  13 root     system        4096 Oct 06 03:05 .
drwxr-xr-x   3 root     system         256 Oct 05 17:38 ..
-rw-r-----   1 root     system       13154 Jun 28 14:19 A_README
drwxrwxr-x   2 root     system        4096 Oct 11 03:05 CS_input_source
drwxrwxr-x   3 root     system         256 Oct 05 17:38 accounting
drwxrwxr-x   2 root     system        4096 Oct 05 17:38 bin
drwxrwxr-x   2 root     system        4096 Oct 11 03:05 data
drwxrwxr-x   4 root     system        4096 Jun 28 14:20 description
drwxrwxr-x   2 root     system        4096 Oct 05 17:38 etc
drwxrwxr-x   2 root     system        4096 Jun 28 14:20 examples
drwxrwxr-x   3 root     system         256 Jun 28 14:20 help
drwxrwxr-x   2 root     system        4096 Oct 11 01:05 history
drwxrwxr-x   2 root     system         256 Oct 06 03:05 log
drwxrwxr-x   8 root     system         256 Jun 28 14:20 scripts
```

Figure 5-7 List of files in the installation directory for an AIX remote deployment

3. Review the crontabs directory, verifying that the root file has been updated to include the scheduling of the Tivoli Usage and Accounting Manager tasks. The command we used and the results are shown in Figure 5-8.

```
# tail -7 /var/spool/cron/crontabs/root
#
# TUAM UNIX/Linux Data Collector scripts
#
5 1 * * * (/opt/ibm/tuam/collectors/Unix/etc/ituam_uc_nightly 1>
/opt/ibm/tuam/collectors/Unix/log/ituam_uc_nightly.log 2>&1)
3,13,23,33,43,53 * * * * /opt/ibm/tuam/collectors/Unix/etc/check_pacct
5 3 * * *
(/opt/ibm/tuam/collectors/Unix/scripts/enterprise/CS_nightly_consolidation
1> /opt/ibm/tuam/collectors/Unix/log/CS_nightly_consolidation.log 2>&1)
#45 3 * * * (/opt/ibm/tuam/collectors/Unix/scripts/enterprise/CS_send 1>
/opt/ibm/tuam/collectors/Unix/log/CS_send.log 2>&1)
```

Figure 5-8 Tivoli Usage and Accounting Manager processes added to the crontab

Each of these scheduled tasks will be started by the cron scheduler according to the timing parameters specified in the cron table. The scripts and the cron defaults are described in Table 5-1 on page 136.

Table 5-1 Scheduled scripts and their default execution time

Script name	Default execution	Description
ituam_uc_nightly	01:05 a.m.	Collects raw AIX accounting data and file system usage
check_pacct	Every 10 minutes	Manages the pacct file
CS_nightly_consolidation	03:05 a.m.	Consolidates the nightly accounting and storage files into the CSR file
CS_send (optional)	03:45 a.m.	Transfers the consolidated CSR file to the Application server

The scheduled scripts produce job logs that contain information about the execution of each of the tasks. The scripts also produce data files containing accounting and file system information.

Table 5-2 shows the name of the output log file, and the data file directories on our lpar04 AIX system. The Output log files are located in directory /opt/ibm/tuam/collectors/Unix/log/.

Table 5-2 Output from the scheduled data collector scripts

Script name	Output log files	Output data file directory
ituam_uc_nightly	ituam_uc_nightly.log	../collectors/Unix/accounting/lpar04
check_pacct	-	-
CS_nightly_consolidation	CS_nightly_consolidation.log	../collectors/Unix/CS_input_source
CS_send (optional)	CS_send.log	-

5.2.3 Transferring collected data to the Application server

The consolidated data produced by the CS_nightly_consolidation script is transferred to the Application server for processing. Several techniques can be used to transfer data from the AIX server to the Application server, such as FTP and secure copy (scp).

We elect to use scp for two reasons:

- The ssh software on the processing server and the AIX server has already been installed and configured, and was used for the remote deployment of the data collector. (see “Remote installation of the AIX data collector” on page 129)

- By using the **scp** command on the Application server to pull the data, the copying of data from the remote systems can be centralized on just the processing server.

Manually transferring the data

The data can be manually transferred between the AIX server and the Application server using a telnet session with the Application server as follows:

1. Connect to the Application server and log on as the root user.
2. Change to the /opt/ibm/tuam/processes directory relating to the "Process id" value within the job file that processes the data on the Application server. In our case, the process id value in the job file is "UNIXProcessing." This is the directory used by the job file to process the data. We change to this directory using the following command:
   ```
   # cd /opt/ibm/tuam/processes/UNIXProcessing
   ```
3. Use the secure copy command to "pull" the data from the AIX server to the Application server. The command and results are shown in Figure 5-9.

```
# scp root@lpar04:/opt/ibm/tuam/collectors/Unix/CS_input_source/CS_s
um_20071016.csv /tmp
CS_sum_20071016.csv                                          100%   19KB
19.4KB/s   00:00#
```

Figure 5-9 Results of the secure copy from the AIX server to the Application server

Automating the transfer of data

Most of the time, it is impractical to transfer data manually on a daily basis. We create a new script called CS_pull based on the CS_send script that is provided with Tivoli Usage and Accounting Manager. We use this new script to transfer data from the AIX system to the processing server. The script is modified and run from the processing server. You can use the changes described here as the basis for your own customization. A complete listing of the sample script is available in the appendix under "Sample script to transfer UNIX data" on page 287.

Perform the following steps to create the new CS_pull script:

1. Change to the directory where the CS_send script is located
   ```
   cd /opt/ibm/tuam/collectors/unix/scripts/enterprise
   ```
2. Copy the CS_send script to a new file with the name of CS_pull
   ```
   cp CS_send CS_pull
   ```

3. The CS_pull script uses variables set in the Tivoli Usage and Accounting Manager configuration files. Edit the A_config.par file in the /opt/ibm/tuam/collectors/unix/data directory.

   > **Note:** We recommend that you create a backup of this file before you edit and make changes.

4. Set the values for the following variables in the A_config.par configuration file:

```
CS_PLATFORM=lpar04
CS_USER=root
CS_METHOD=SCP
CS_PROC_PATH=/opt/ibm/tuam/processes
CS_UNIXOS_PROCDIR=UNIXProcessing
```

5. Review the GEN_* variable settings in A_config.par. These settings determine which files are retrieved from the AIX server. In our case, we accept the default settings for these variables.
6. Various commands and processes in the data transfer script need to be updated for the AIX platform, and to perform a pull rather than a push operation.
   a. Update the ORIGIN variable to match the directory structure on the remote system. Because we use remote deployment of the UNIX data collector, the directory name will be consistent and we hard code the value. We set it as follows:
      `ORIGIN="/opt/ibm/tuam/collectors/Unix/CS_input_source"`
   b. Amend the source code that checks for the existence of the files to use the **ssh** command with the **ls** command as parameter. We do this because the listing of the files needs to be performed on the remote machine. We change:

```
if test ! -f ${ORIGIN}/${arg}${DATE}.csv
```

      into

```
file_exists=`ssh ${CS_USER}@${CS_PLATFORM} ls \
   ${ORIGIN}/${arg}${DATE}.csv`
file_status=$?
if test "$file_status" -gt "0"
```

   c. In the SCP data transfer section of the script identified by the text:

```
elif [ "${XFER}" = "SCP" ]
```

      update the status message to reflect that the script performs a pull rather than a push.

The destination for the retrieved file is also updated. We change the status message of:

```
echo "    Sending $ORIG_FILE to
$DESTIN\\$SENDER_PLATFORM\\$FILE_DATE.txt"
```

into

```
echo "    Retrieving $ORIG_FILE to
${CS_PROC_PATH}/${DESTIN}/${CS_PLATFORM}/${file_item}"
```

d. We assume that the destination directory for the retrieved files has been created. Therefore we remove the creation of the destination directory. We delete this code:

```
su - ${ITUAM_USER} -c "ssh -l ${CS_USER} ${CS_PLATFORM} \"cmd /c
mkdir ${CS_PROC_PATH}\\${DESTIN}\\${SENDER_PLATFORM}\"" 2>
/dev/null 1>&2
```

e. We replace the **scp** command so that it retrieves data from the remote server. Remove the source code in:

```
su - ${ITUAM_USER} -c "scp ${ORIG_FILE}
${CS_USER}@${CS_PLATFORM}:\"${CS_PROC_PATH}\\${DESTIN}\\${SENDER_
PLATFORM}\\${FILE_DATE}.txt\"" 1>
${ITUAM_HISTORY}/tmp_CS_send_scp_${DATE}.log 2>&1
```

and replace it with:

```
su - ${ITUAM_USER} -c "scp ${CS_USER}@${CS_PLATFORM}:${ORIG_FILE}
${CS_PROC_PATH}/${DESTIN}/${CS_PLATFORM}/${file_item}" 1>
${ITUAM_HISTORY}/tmp_CS_pull_${CS_PLATFORM}_scp_${DATE}.log 2>&1
```

7. Other non-essential changes that we recommend:
   - Replace the `CS_send` characters with `CS_pull`. One of the benefits of this is that stdout and stderr data is routed to a file with a more appropriate name and that messages reflect the correct script name.
   - Add a test for the existence of the destination directory.
   - Update the script so that it can be used for multiple servers (whose values are assigned to the CS_PLATFORM) variable.
   - Update messages to reflect that files are being retrieved rather than sent.

Executing the data transfer script to retrieve data

We set up the directory structure and schedule the data transfer script for execution on a daily basis.

1. First we create the directory structures for the retrieved files. These new directories are located off the /opt/ibm/tuam/processes directory. The directory structure we create matches the “Process id” value within the job file

that processes the data on the Application server. In our case the two "Process id" values are "UNIXProcessing" and "UnixFS". The data we are retrieving is from lpar04. The commands to create the directories are shown in Example 5-2.

Example 5-2 Create the directories for data retrieved from the remote system

```
cd /opt/ibm/tuam/processes
mkdir UNIXProcessing
mkdir UNIXProcessing/lpar04
mkdir UnixFS
mkdir UnixFS/lpar04
```

2. Update the cron table for the root user using the **`crontab -e`** command. Add the line that schedules the CS_pull script as shown in Figure 5-3. Adjust the minute and hour setting, and output log file accordingly.

Example 5-3 Add the scheduling of the data transfer script to the cron table

```
45 3 * * * (/opt/ibm/tuam/collectors/unix/scripts/enterprise/CS_pull 1>
/opt/ibm/tuam/collectors/unix/log/CS_pull_lpar04.log 2>&1)
```

5.2.4 Loading the AIX data into Tivoli Usage and Accounting Manager

Once the AIX data has been transferred to the processing server, it can be loaded into the Tivoli Usage and Accounting Manager database. Using the ISC we create and execute a new Job Runner job file which contains the necessary steps to perform a load of the UNIX data to the database.

The overall processing of the UNIX data is shown in Figure 5-10 on page 141. We created this in a file called LoadUNIX.xml in the jobfiles directory for running it with Job Runner.

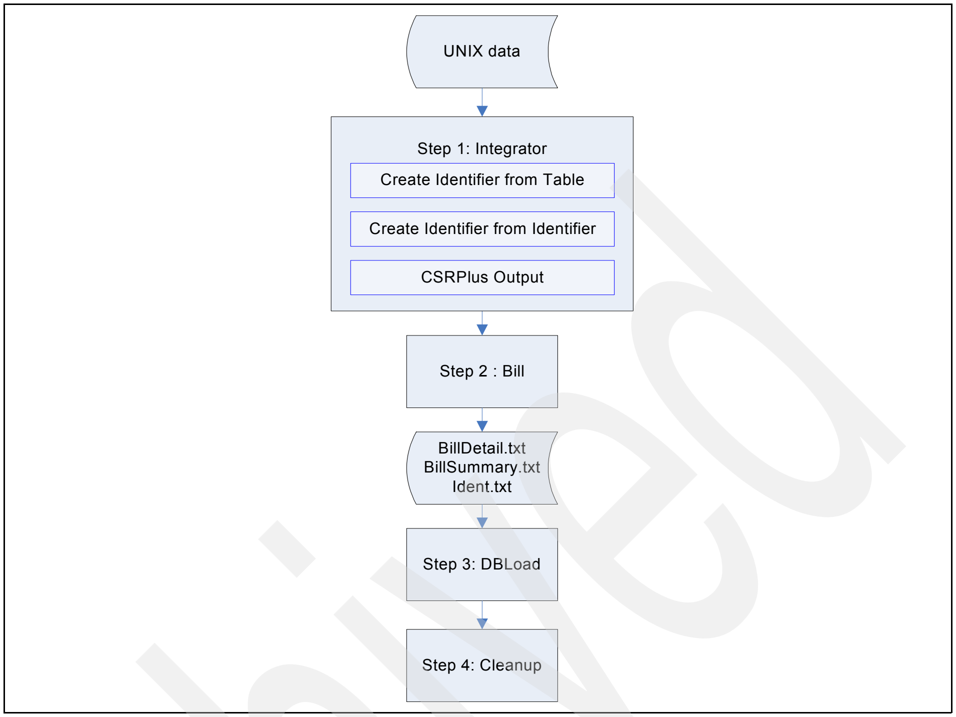


Figure 5-10 UNIX data processing

We do not discuss the structure of the job itself here; more information regarding Job Runner is provided in chapter 3. The processing steps performed with the LoadUNIX.xml follow.

1. The first step is the integrator. It loads CSR data that is retrieved by the CS_pull program described previously. The input stage of the step is shown in Example 5-4. It identifies the input files for UNIX processing and UNIX file system (UNIXFS).

Example 5-4 Integrator input stage

```
<Step id="Integrator" type="ConvertToCSR" programType="integrator" programName="integrator">
  <Integrator>
    <Input name="CSRInput" active="true">
      <Files>
      <File name="%ProcessFolder%/UNIXProcessing/lpar04/CS_sum_%LogDate_End%.csv" />
      <File name="%ProcessFolder%/UnixFS/lpar04/CS_sum_fs_%LogDate_End%.csv" />
      </Files>
    </Input>
  </Integrator>
</Step>
```

2. The CreateIdentifierFromTable stage retrieves account codes from the Accttabl.txt file based on the SYSTEM_ID value within the input file. This stage is shown in Example 5-5.

Example 5-5 Add the text for the first stage of the integrator step

```
<Stage name="CreateIdentifierFromTable" active="true">
  <Identifiers>
    <Identifier name="Account_Code_TMP">
      <FromIdentifiers>
        <FromIdentifier name="SYSTEM_ID" offset="1" length="10"/>
      </FromIdentifiers>
    </Identifier>
  </Identifiers>
  <Files>
    <File name="/opt/ibm/tuam/processes/Accttabl.txt" type="table"/>
    <File name="UNIXException.txt" type="exception" format="CSROutput"/>
  </Files>
  <Parameters>
    <Parameter exceptionProcess="true"/>
    <Parameter sort="true"/>
    <Parameter upperCase="false"/>
    <Parameter writeNoMatch="false"/>
    <Parameter modifyIfExists="true"/>
  </Parameters>
</Stage>
```

3. The CreateIdentifierFromIdentifier stage appends the host_name to the account code. The definition is shown in Example 5-6.

Example 5-6 Add the second stage to the integrator step

```
<!--  add hostname as last part of the account code -->
<Stage name="CreateIdentifierFromIdentifiers" active="true">
  <Identifiers>
    <Identifier name="Account_Code">
      <FromIdentifiers>
        <FromIdentifier name="Account_Code_TMP" offset="1" length="40"/>
        <FromIdentifier name="SYSTEM_ID" offset="1" length="20"/>
      </FromIdentifiers>
    </Identifier>
  </Identifiers>
  <Parameters>
    <Parameter modifyIfExists="true"/>
    <Parameter keepLength="true"/>
  </Parameters>
</Stage>
```

4. The output from the Integrator step is defined in a CSROutput stage. The output file is called AcctCSR.txt as shown in Example 5-7.

Example 5-7 Add the third stage to the integrator step

```
<Stage name="CSRPlusOutput" active="true">
  <Files>
    <File name="AcctCSR.txt" />
  </Files>
</Stage>
```

5. The next step is a process step. This step invokes the billing program (called Bill). Three files are produced as output from the billing step, namely:
   – BillDetail.txt
   – BillSummary.txt
   – Ident.txt

   This step is shown in Example 5-8.

Example 5-8 Add the billing step after the integrator step

```
<Step   id="Bill"
        description="Standard Processing for UNIX"
        type="Process"
        programName="Bill"
        programType="java"
        active="true">
  <Bill>
    <Parameters>
    </Parameters>
  </Bill>
</Step>
```

6. After billing information is created, the data is ready for loading to the database. Example 5-9 shows the DBLoad program to load data to the Tivoli Usage and Accounting Manager database.

Example 5-9 Add the DatabaseLoad step

```
<Step   id="DatabaseLoad"
        description="Database Load for UNIX"
        type="Process"
        programName="DBLoad"
        programType="java"
        active="true">
  <DBLoad>
    <Parameters>
```

```
    </Parameters>
  </DBLoad>
</Step>
```

7. The final step is for cleaning up log files. The definition in Example 5-10 invokes the Cleanup program to remove data files that have been kept for longer than 45 days.

Example 5-10 Add the Cleanup step

```
<Step   id="Cleanup"
        description="Cleanup UNIX"
        type="Process"
        programName="Cleanup"
        programType="java"
        active="false">
  <Parameters>
    <Parameter DaysToRetainFiles="45"/>
  </Parameters>
</Step>
```

To load the data into the database run the job you have created. Perform this task manually, or schedule the job to run automatically.

Manually executing the job

Execute the job manually using the ISC.

1. Using the ISC console, select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files** and select the LoadUNIX.xml file.
2. Click the **Run Job** button to execute the job.
3. The Job Runner parameters screen is displayed as shown in Figure 5-11 on page 145. Click **OK** to execute the job.

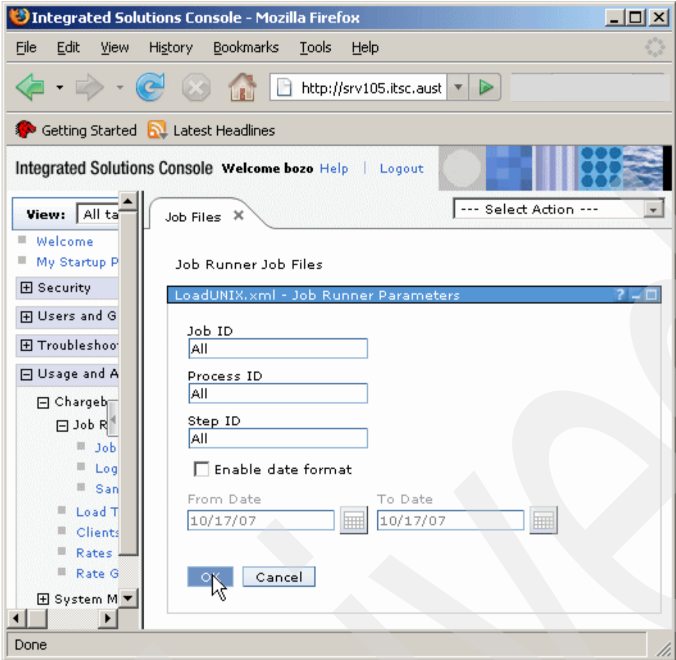


Figure 5-11 Job Runner parameter window for the LoadUNIX job

4. An information message is displayed with the results of the job execution.

Scheduling the job to run automatically

Use the cron scheduler to run the job automatically.

1. Log on to the processing server as root.
2. Enter the **`crontab -e`** command to edit the cron table.
3. Add the line that schedules the LoadUNIX.xml job as shown in Figure 5-12. Adjust the minute and hour setting, and the output log file accordingly. In our case, the LoadUNIX job runs at 6:17 a.m. every morning.

```
17 6 * * * (/opt/ibm/tuam/bin/startJobRunner.sh LoadUnix.xml >>
/opt/ibm/tuam/logs/jobrunner/LoadUnix.log 2>&1)
```

Figure 5-12 Add the LoadUNIX job to the cron table

4. Save the updated cron table and exit the editor.

5.2.5 Reviewing the results of the executed job

The results of the job can be viewed using the ISC as follows:

1. Select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Log Files**. A list of executed jobs is displayed as in Figure 5-13. Note that the green check mark in the main window indicates a successful completion of the job. A red cross indicates a job failure.

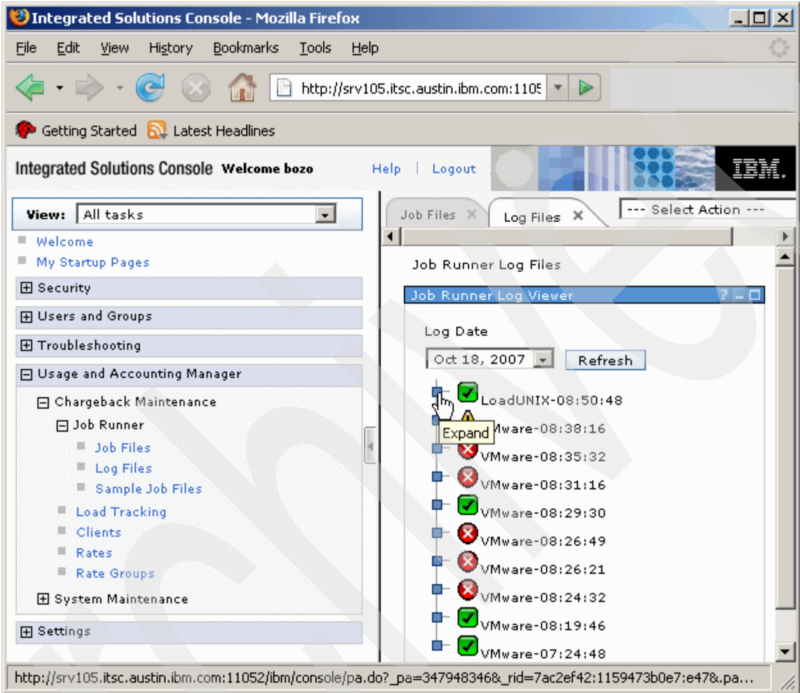


Figure 5-13 Expand the log files view for the LoadUNIX job

2. In the main window of the ISC, select the blue box on the left side of the “LoadUNIX” job to expand the log view.
3. In the main window of the ISC select the blue box next to the UNIXProcessing process to expand the log file view further. The three steps that executed are displayed (see Figure 5-14 on page 147). The Cleanup step is not executed because the step includes a parameter of `active=false`. If you want the Cleanup step to be executed set `active=true`.

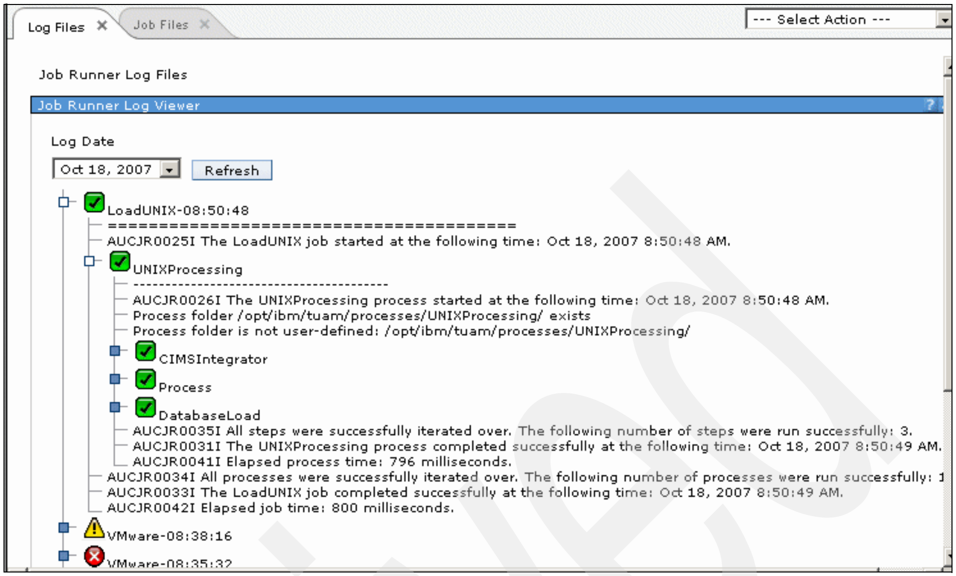


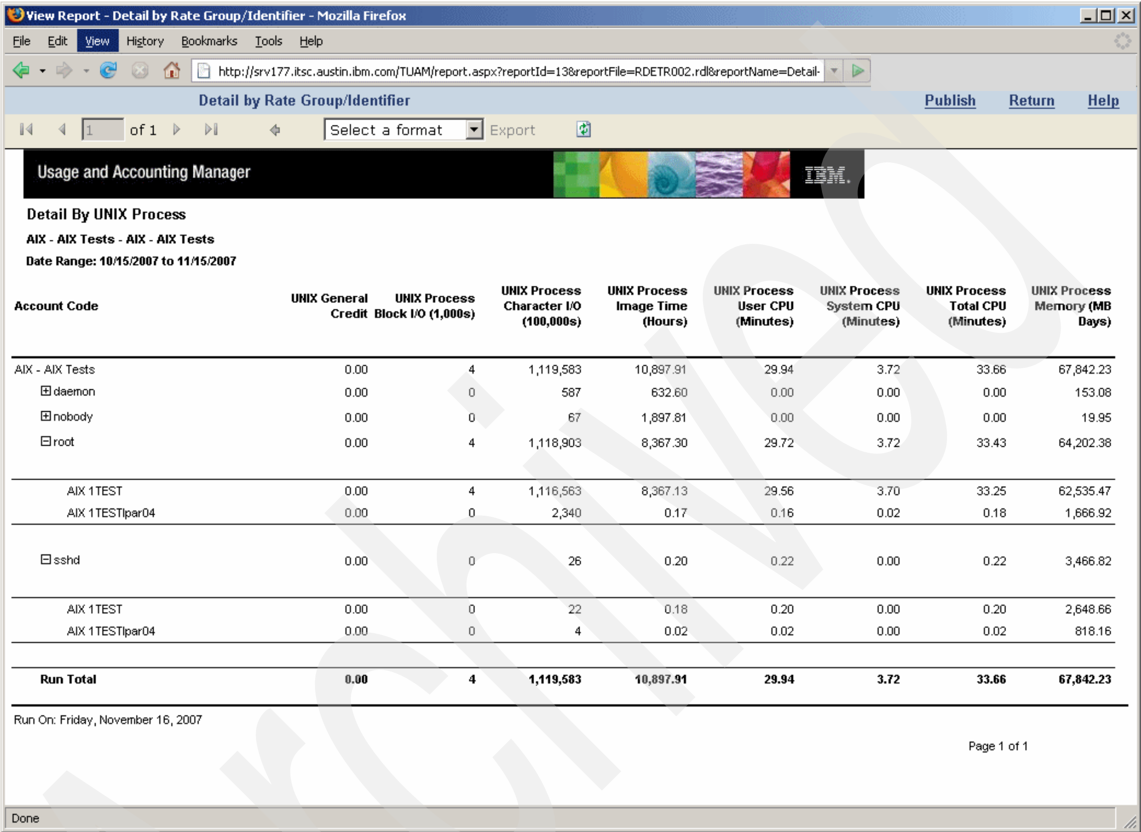
Figure 5-14 Expand the log view to show the steps in the LoadUNIX job

4. Select each of the blue filled boxes beside each step to expand the view. This displays the complete log output for the step. In particular, examine the DatabaseLoad step to review the results of the database update. Our results are shown in Figure 5-15.

Figure 5-15 Results of the DatabaseLoad step of the LoadUNIX job

5.2.6 UNIX usage data reports

Figure 5-16 shows a report for UNIX usage data. We generated this report using the Windows Reporting Service on `http://srv177/tuam/`.



Detail By UNIX Process

AIX - AIX Tests - AIX - AIX Tests

Date Range: 10/15/2007 to 11/15/2007

Account Code	UNIX General Credit	UNIX Process Block I/O (1,000s)	UNIX Process Character I/O (100,000s)	UNIX Process Image Time (Hours)	UNIX Process User CPU (Minutes)	UNIX Process System CPU (Minutes)	UNIX Process Total CPU (Minutes)	UNIX Process Memory (MB Days)
AIX - AIX Tests	0.00	4	1,119,583	10,897.91	29.94	3.72	33.66	67,842.23
⊞ daemon	0.00	0	587	632.60	0.00	0.00	0.00	153.08
⊞ nobody	0.00	0	67	1,897.81	0.00	0.00	0.00	19.95
⊟ root	0.00	4	1,118,903	8,367.30	29.72	3.72	33.43	64,202.38
AIX 1TEST	0.00	4	1,116,563	8,367.13	29.56	3.70	33.25	62,535.47
AIX 1TESTlpar04	0.00	0	2,340	0.17	0.16	0.02	0.18	1,666.92
⊟ sshd	0.00	0	26	0.20	0.22	0.00	0.22	3,466.82
AIX 1TEST	0.00	0	22	0.18	0.20	0.00	0.20	2,648.66
AIX 1TESTlpar04	0.00	0	4	0.02	0.02	0.00	0.02	818.16
Run Total	**0.00**	**4**	**1,119,583**	**10,897.91**	**29.94**	**3.72**	**33.66**	**67,842.23**

Run On: Friday, November 16, 2007

Page 1 of 1

Figure 5-16 Report on UNIX process data

5.3 Windows data collection

This section discusses the collection and processing of the Windows data. The topics covered are:

- 5.3.1, “Installation of Windows process data collector” on page 149
- 5.3.2, “Verifying the Windows Process data collector installation” on page 151
- 5.3.3, “Windows process data files” on page 153
- 5.3.4, “Loading Windows data” on page 154

5.3.1 Installation of Windows process data collector

The Windows process data collector is installed as follows:

1. Manually install the Windows Process data collector by executing the setup-tuam-wpc-7-1-0-windows_32_64.exe file. Make sure that the setup.jar and wpc.rsp files are located in the directory from which you are running the installer as shown in Figure 5-17.

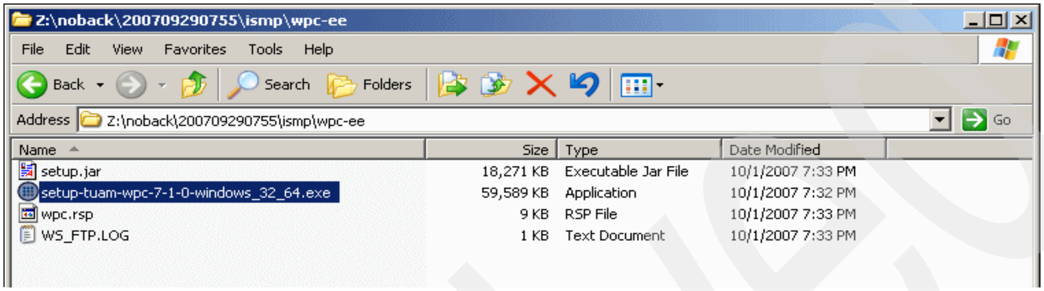


Figure 5-17 Manually execute the Windows Process data collector installer

2. If a security warning is displayed click **Run**.
3. The install wizard starts. Click **Next** at the welcome window.
4. Accept the license agreement and click **Next**.
5. If required, modify the directory name of the installation path and click **Next** (Figure 5-18.)

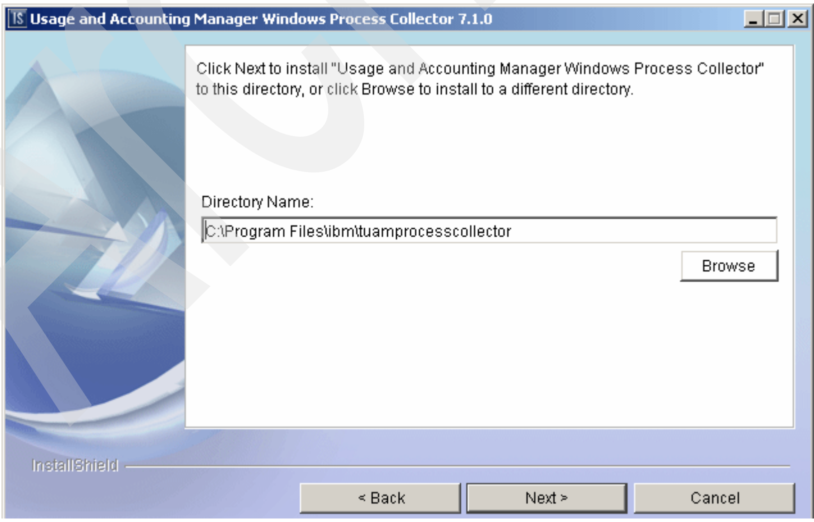


Figure 5-18 Set the directory path for the Windows Process data collector

6. If necessary, update the data collector configuration according to your requirements as shown in Figure 5-19. We accept the defaults and click **Next**. The option to Start application after installation and during reboot allows the job to run automatically. Consequently, there is no possibility to schedule the Windows data collection as we do on AIX.

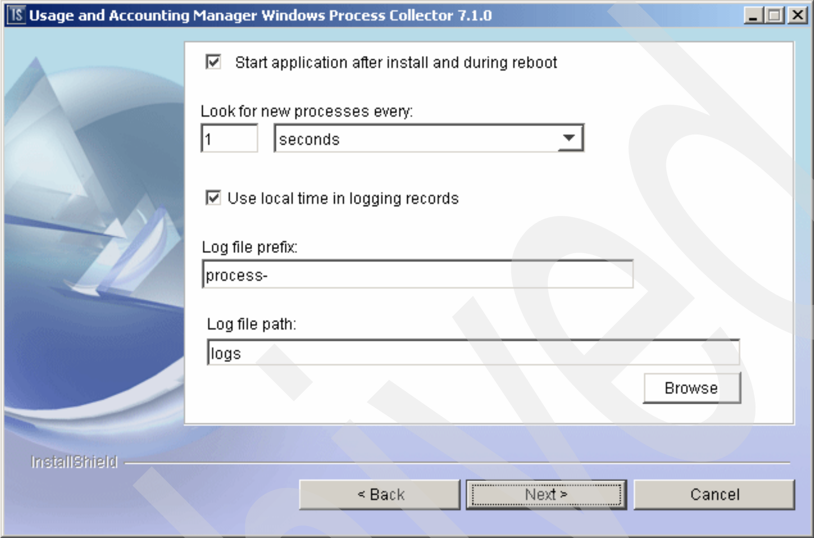


Figure 5-19 Customize the Windows Process data collection

7. Review the summary information and click **Install**.
8. The installation progress window is displayed. Review the information in the summary information window and click **Finish** to complete the installation. Figure 5-20 on page 151 shows the successful completion window.

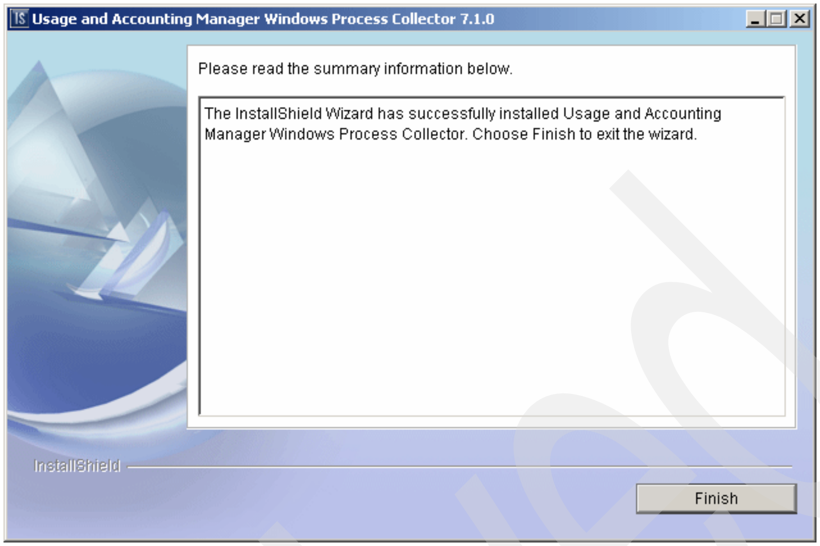


Figure 5-20 Successful installation of the Windows Process data collector

5.3.2 Verifying the Windows Process data collector installation

Three techniques are used to verify the deployment, namely a directory listing, a listing of the services installed, and a display of an executing task. You perform these from the system where the Windows Process data collector was installed.

- List the contents of the directory that contains the data collector software. Use the directory path specified during the install (shown in Figure 5-18 on page 149.) The files located in the directory are listed in Figure 5-21 on page 152.

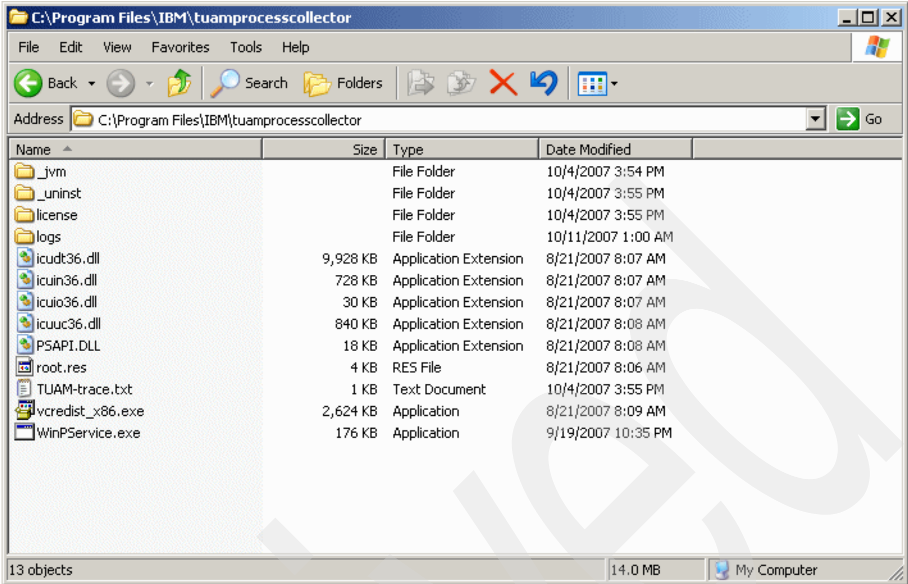


Figure 5-21 Directory listing of the Windows Process data collector install path

- Using the Windows menus select **Control Panel** → **Administrative Tools** → **Services.** Confirm that the Usage and Accounting Manager Process Collector has been added as a service as shown in Figure 5-22.

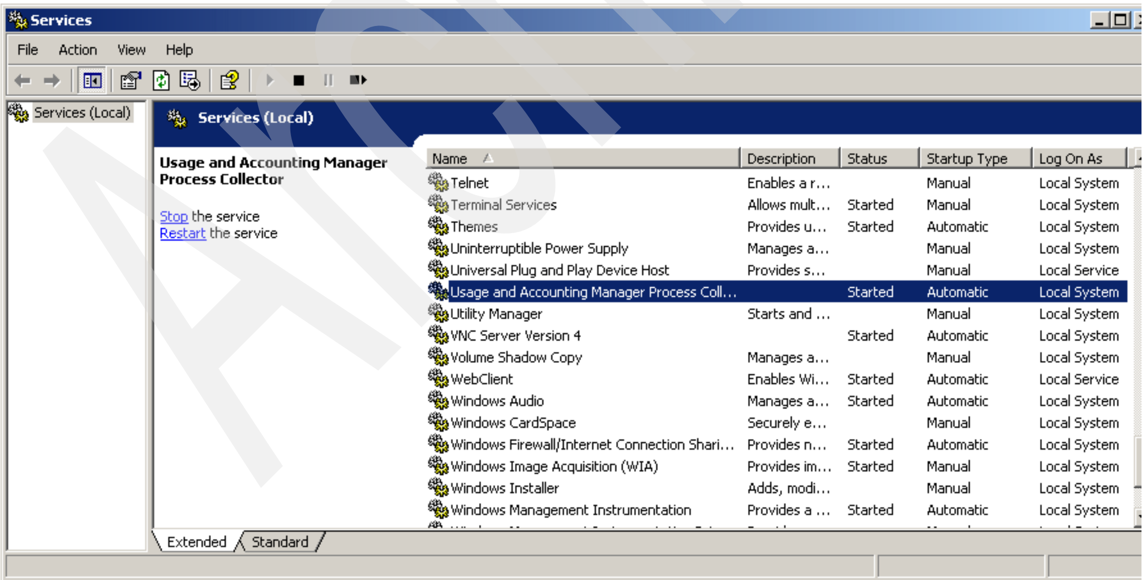


Figure 5-22 Listing of the Usage and Accounting Manager Process Collector service

- Start the Windows Task Manager and select the Processes tab. Verify that the WINPService.exe task is running as shown in Figure 5-23.

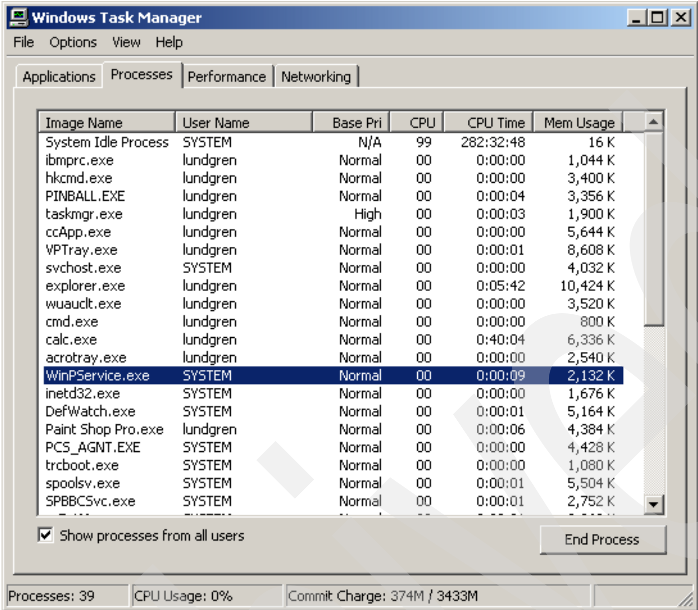


Figure 5-23 The executing Usage and Accounting Manager Process Collector service

5.3.3 Windows process data files

The results of the collector are data files. These data files are located based on the Log file path parameter that is specified at installation time (see Figure 5-19 on page 150). A sample of a file list created by the Windows data collector is displayed in Figure 5-24 on page 154.

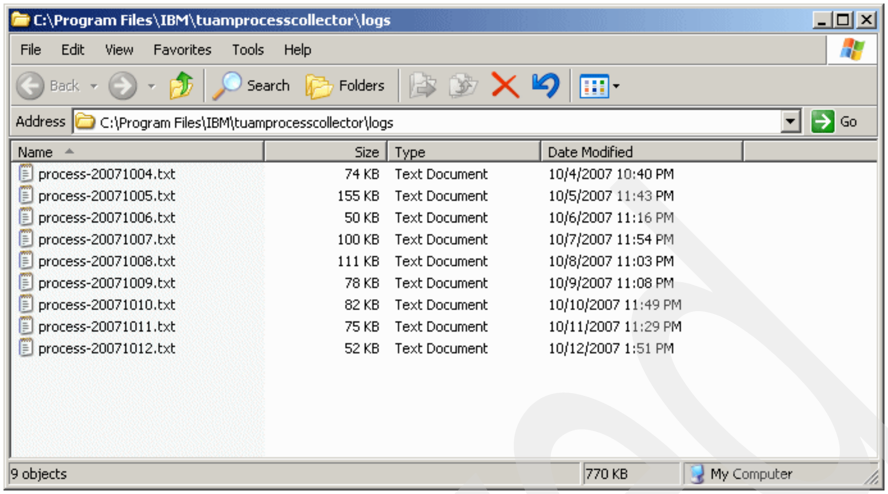


Figure 5-24 List of files created by the Windows data collector

The files produced by the Windows data collector must be transferred to the processing server. Each of these daily files should be transferred after midnight on the day it was produced because the file is switched at midnight. Use the technique most suited to your environment to perform the transfer. If the Windows machines run an ssh protocol server (such as OpenSSH), you can use the transfer script which runs on the Linux server as described in 5.2.3, “Transferring collected data to the Application server”.

5.3.4 Loading Windows data

The SampleWinProcess.xml job supplied with Tivoli Usage and Accounting Manager uses the WinProcess.wsf. The WinProcess.wsf is a Windows script file which converts the input data to CSR format. This script file will not work in our example scenario because the processing server is installed on Linux (and the Windows script file will only execute on a Windows platform.)

The WinProcess.wsf script performs the following actions:

- Extract type “S” (start) and type “I” (interval) records.
- Remove data that is not required for accounting purposes.
- Format the data into CSR format for output using input fields depending on the record type (start or interval).

We use the functionality within Tivoli Usage and Accounting Manager to reproduce the processes performed by the WinProcess.wsf script. An overview of the steps, and stages involved, is shown in Figure 5-25 on page 155. The complete listing of the job file is provided in the appendix, under “Sample job to load Windows process data” on page 312.

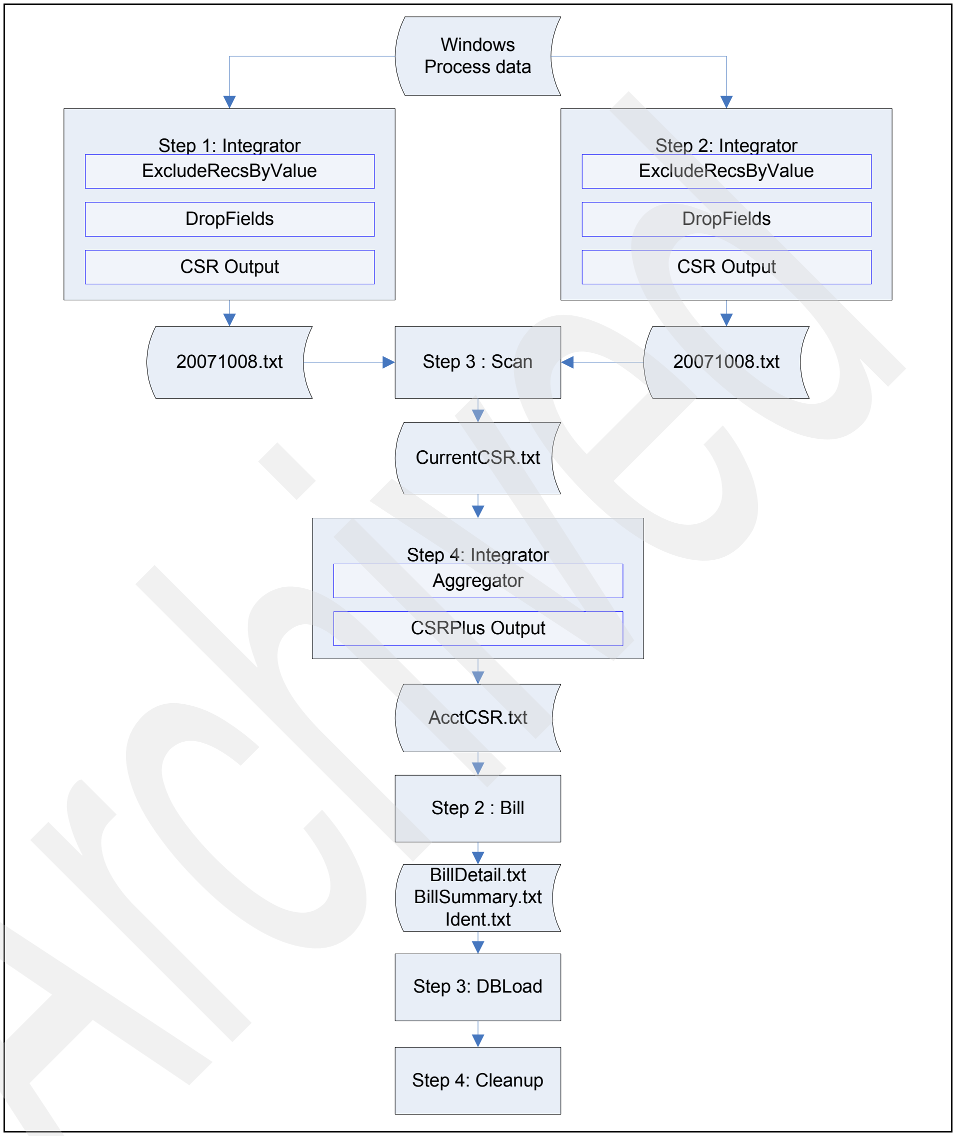


Figure 5-25 Overview of the Windows processing to replace the WinProcess.wsf script

Four steps are required to reproduce the functionality in the WinProcess.wsf script. These steps are described in Table 5-3 on page 156.

Table 5-3 Summary description of the steps that replace the WinProcess.wsf script.

Step	Stage	Description	Output
Integrator1	ExcludeRecsByValue	Extract the “start” records	
	DropFields	Remove the fields not required for accounting	
	CSROutput	Produce the CSR file in the “start” directory	20071004.txt
Integrator2	ExcludeRecsByValue	Extract the “interval” records	
	DropFields	Remove the fields not required for accounting	
	CSROutput	Produce the CSR file in the “interval” directory	20071004.txt
Scan		Identify files matching the date criteria	CurrentCSR.txt
Integrator3	Aggregator	Aggregate the merged data	

The output from the aggregator is used in the steps that follow it in the job. In our case, the final integrator step contains further stages where the account code is added using the CreateIdentifierFromIdentifiers function and the AcctCSR.txt file is produced. Thereafter the data will be loaded into the database using Bill and DBLoad, and the Cleanup is performed similar to the UNIX processing described in 5.2.4, “Loading the AIX data into Tivoli Usage and Accounting Manager” on page 140.

Sample code to perform each of the functions in Table 5-3 is presented in the following paragraphs. Because the Integrator1 and Integrator2 steps are similar, only one of these steps is documented here. Note that the Input section of both integrator step 1 and 2 refers to the same file.

- Integrator1: ExcludeRecsByValue

  This stage uses the ExcludeRecsByValue function of the integrator to exclude all but the start records (“S” in the RecordType field). See Example 5-11 on page 157. We also exclude records that have no parents (System and System Idle Process). This process also maps the header start and end time to the Process start time.

  The second integrator step processes the I records and maps the header start and end time with the Interval start and end time.

Example 5-11 Sample stage to exclude records by value

```
<Stage active="true" name="ExcludeRecsByValue">
  <Identifiers>
    <Identifier name="RecordType" cond="EQ" value="RecordType"></Identifier>
    <Identifier name="RecordType" cond="EQ" value="E"></Identifier>
    <Identifier name="RecordType" cond="EQ" value="I"></Identifier>
    <Identifier name="ParentProcessId" cond="EQ" value="0"></Identifier>
  </Identifiers>
</Stage>
```

- Integrator1: DropFields

  This stage utilizes the DropFields function of the integrator to remove the specified fields from the record. In this case, the ParentProcessId and the RecordType fields are dropped because we no longer need them. Example 5-12 shows the source code for this stage.

Example 5-12 Sample stage to drop fields from the output file

```
<Stage active="true" name="DropFields">
  <Fields>
    <Field name="ParentProcessId"></Field>
    <Field name="RecordType"></Field>
  </Fields>
</Stage>
```

- Integrator1: CSROutput

  This defines the output file for this step (Example 5-13). Later in the job, the scan step retrieves the file and merges it with the output from the other integrator step.

Example 5-13 Sample stage to write the CSR data to a file

```
<Stage active="true" name="CSROutput">
  <Files>
    <File name="%ProcessFolder%\S\20071018.txt"></File>
  </Files>
</Stage>
```

- Scan

  This step merges the file and concatenates it with other similarly named files, and the output is placed in the process definition directory with a name of CurrentCSR.txt. Example 5-14 is an example of this code.

Example 5-14 Sample step to scan for files to be processed further

```
<Step id="Scan" description="Scan LoadWinProcess" type="Process"
programName="Scan" programType="java" active="true">
```

```
  <Parameters>
    <Parameter retainFileDate="false"></Parameter>
    <Parameter allowMissingFiles="true"></Parameter>
    <Parameter allowEmptyFiles="true"></Parameter>
    <Parameter useStepFiles="false"></Parameter>
  </Parameters>
</Step>
```

- Aggregator

  This step uses the Aggregator function as a stage within an integrator step. We use it to aggregate the file according to the identifiers for which the specified resources are summed. This is shown in Example 5-15.

Example 5-15 Sample stage to aggregate the Windows metrics accordingly

```
<Stage name="Aggregator" active="true" trace="false">
  <Identifiers>
    <Identifier name="Feed"></Identifier>
    <Identifier name="ProcessName"></Identifier>
    <Identifier name="ProcessPath"></Identifier>
    <Identifier name="Server"></Identifier>
    <Identifier name="User"></Identifier>
    <Identifier name="PriorityClass"></Identifier>
    <Identifier name="BasePriority"></Identifier>
  </Identifiers>
  <Resources>
    <Resource name="WINELPTM"></Resource>
    <Resource name="WINCPUTM"></Resource>
    <Resource name="WINKCPUT"></Resource>
    <Resource name="WINCPUUS"></Resource>
    <Resource name="WINRDREQ"></Resource>
    <Resource name="WINKBYTR"></Resource>
    <Resource name="WINWRREQ"></Resource>
    <Resource name="WINKBWRI"></Resource>
  </Resources>
  <Parameters>
    <Parameter defaultAggregation="false"></Parameter>
  </Parameters>
</Stage>
```

The running of this job is similar to the UNIX processing discussed in 5.2.5, “Reviewing the results of the executed job” on page 146.

6

Virtualized environment accounting

Running multiple systems in a shared hardware environment does require the ability to track resource usages and accounting for charge back. This chapter focuses on two common environments:

- 6.1, “System p virtualization and AIX Advanced Accounting” on page 160
- 6.2, “Virtualization with VMware” on page 182

6.1 System p virtualization and AIX Advanced Accounting

In addition to the standard UNIX collector, Tivoli Usage and Accounting Manager offers the data collection from AIX Advanced Accounting, including the Virtual I/O server (VIOS). This section discusses data collection on IBM System p servers and the Virtual I/O server, including the following topics:

6.1.1 Architecture for AIX Advanced Accounting scenario

Figure 6-1 shows the scenario for the data collection in a System p virtualization environment.

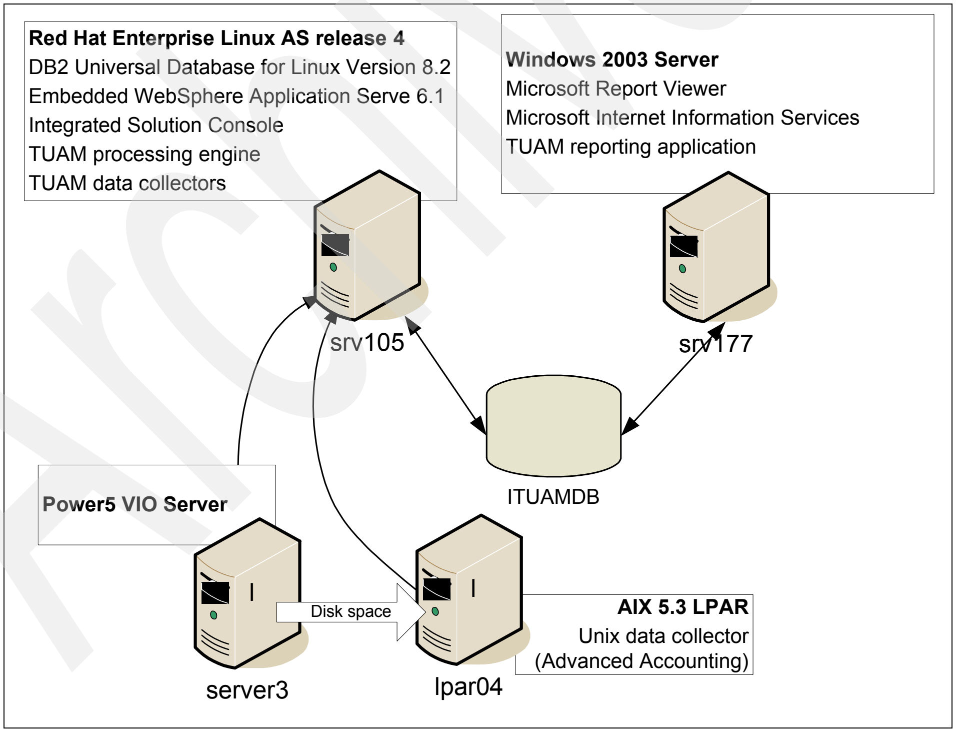


Figure 6-1 Virtualization with AIX Advanced Accounting and Virtual I/O Server

The Virtual I/O Server (VIOS) is providing the disk space for our AIX server lpar04. AIX Advanced Accounting is used on VIOS server and AIX partitions to collect data on AIX usage and VIOS disk I/O. The accounting files are:

aacct_1	Process records
aacct_4	Environmental data (memory, entitlement, and so on.)
aacct_6	File system activity
aacct_7	Network interface I/O
aacct_8	Disk I/O
aacct_10	VIOS server I/O
aacct_11	VIOS client I/O

6.1.2 Virtual I/O server data collection

The Virtual I/O server data collector is delivered within the Server itself because the machine has a closed architecture; it cannot be installed separately. Virtual I/O Server V1.4 fix pack 8.1.0 or later is recommended to run Tivoli Usage and Accounting Manager agents. At a high level, the following steps are performed to collect and use VIOS data:

- "Enable the Virtual I/O server data collector" on page 161
- "Prepare for transferring VIOS data" on page 162
- "Preparing data processing for VIOS data" on page 163
- "Processing and loading VIOS data" on page 164

Enable the Virtual I/O server data collector

The Virtual I/O server runs the AIX Advanced Accounting, which has an extension for VIOS-relevant data gathering. Administration in the Virtual I/O server is typically performed by padmin user, not by root. Figure 6-2 shows how the padmin user activates Tivoli Usage and Accounting Manager data collector.

```
$ lssvc
ITM_base
ITM_premium
TSM_base
ITUAM_base
$ cfgsvc -ls ITUAM_base
ACCT_DATA0
ACCT_DATA1
ISYSTEM
IPROCESS
$ cfgsvc ITUAM_base -attr ACCT_DATA0=10 ACCT_DATA1=10 ISYSTEM=5 IPROCESS=5
$ startsvc ITUAM_base
```

Figure 6-2 Configure and start data collector on the VIOS server

For complete documentation about configuration on the VIOS server refer to the IBM product manual *System i and System p Using the Virtual I/O Server*, available from:

`http://publib.boulder.ibm.com/infocenter/systems/scope/hw/topic/iphb1/iphb1pdf.pdf`

> **Restriction:** The first data collection must be performed on the following day because the final setup will be completed during the nightly script execution.

Prepare for transferring VIOS data

For transferring VIOS data files, we need to set up the destination path and modify two XML files on the Tivoli Usage and Accounting Manager processing server using these steps:

1. Make the directory for your data collection; the path is structured as <%CollectorLogs% directory>/VIOS/<vio hostname> such as:

   ```
   mkdir /opt/ibm/tuam/logs/collectors/VIOS/server3
   ```

2. Copy the sample processing files from the sample job directory. The files are called SampleSecureGetVIOS.xml and SampleSecureGetVIOSManifest.xml.

   ```
   cd /opt/ibm/tuam/jobfiles
   cp ../samples/jobfiles/SampleSecureGetVIOS.xml TransferVIO.xml
   cp ../samples/jobfiles/SampleSecureGetVIOSManifest.xml
   ../collectors/unixlinux/TransferVIOManifest.xml
   ```

3. The TransferVIO.xml is used to run the Remote Product Deployment (RPD) function for transferring accounting data files. We need to update the parameter section for our environment as shown in Example 6-1.

Example 6-1 VIOS parameter in the job file

```
<Parameters>
   <Parameter Host="server3"/>
   <Parameter UserId="root"/>
   <Parameter Password="********"/>
   <Parameter Manifest="TransferVIOManifest.xml"/>
   <Parameter
   RPDParameters="client_CS_path=/opt/IBM/tivoli/ituam/collectors/Unix/
   CS_input_source;CollectorLogs_dir=%CollectorLogs%/VIOS;LogDate=%LogD
   ate_End%;client_name=server3;"/>
   <Parameter Verbose="true"/>
   <Parameter SourcePath="%HomePath%/collectors/unixlinux/"/>
</Parameters>
```

The meanings of the parameters are:

Host	DNS name or IP address of the VIOS server
Manifest	Name of the RPD action definition
client_CS_path	Path of the aacct files on the VIOS server
CollectorLogs_dir	Base directory for VIOS logs on the Tivoli Usage and Accounting Manager server
client_name	Name for the subdirectory "Feed" on the Tivoli Usage and Accounting Manager server
SourcePath	Path where the manifest file is located

> **Restriction:** VIOS user padmin is not allowed to do **sftp**, which is required by this job. Therefore, we are using root for the data transfer.

4. Modify the TransferVIOManifest.xml file because the RPD actions and file definitions are stored there.

   The localpath must be updated for the Tivoli Usage and Accounting Manager application server installation as shown in Example 6-2. Add an action for the aacct10 file, containing the VIOS data.

Example 6-2 VIOS parameter of new action in the manifest for the data transfer

```
<Action name="step_AACCT_10_%client_name%" displayMessage="Getting
nightly AACCT_10 file for %client_name%" actionType="FileGet">
 <Parameters>
   <Parameter name="localpath"
   value="%CollectorLogs_dir%/%client_name%"/>
   <Parameter name="remotefilename"
   value="%client_CS_path%/aacct10_%LogDate%.txt"/>
 </Parameters>
</Action>
```

5. We remove all comments (<!-- -->) to collect all data files.

> **Tip:** If the path for the collector logs does not exists, all your data is stored in one file using the client_name. Remove the file and create a directory instead.

Preparing data processing for VIOS data

The VIOS data is loaded with the advanced accounting collector. We copy the SampleSecureGetVIOS.xml to LoadVIO.xml file:

```
cd /opt/ibm/tuam/jobfiles
```

```
cp ../samples/jobfiles/SampleSecureGetVIOS.xml LoadVIO.xml
```

Because you have set up the TransferVIO.xml for transferring the data, you can remove the first step calling the RPD program and the step dropping fields. Update the path for the file input and ensure that the VIOS data aacct10 is included as in Example 6-3.

Example 6-3 Modifying the input section for VIOS data processing

```
<Input name="AIXAAInput" active="true">
 <Files>
  <File name="%CollectorLogs%/VIOS/server3/aacct1_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/VIOS/server3/aacct4_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/VIOS/server3/aacct6_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/VIOS/server3/aacct7_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/VIOS/server3/aacct8_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/VIOS/server3/aacct10_%LogDate_End%.txt" />
  <File name="%ProcessFolder%/exception.txt" type="exception" />
 </Files>
</Input>
```

Example 6-4 shows the updated path or the CSR output file. The output file is in the CSR plus format.

Example 6-4 Updating the CSR output path for VIOS data processing

```
<Stage name="CSRPlusOutput" active="true">
 <Files>
  <File name="%ProcessFolder%/server3/%LogDate_End%.txt" />
 </Files>
</Stage>
```

Processing and loading VIOS data

Transferring data is shown in Figure 6-3.

```
[root@srv105 bin]# ./startJobRunner.sh TransferVIO.xml
Oct 31, 2007 4:26:28 PM com.ibm.tivoli.tuam.logger.JobLogger startJob
[..]
Oct 31, 2007 4:26:32 PM com.ibm.tivoli.tuam.logger.JobLogger endJob
INFO: AUCJR0033I The TransferVIO job completed successfully at the following
time: Oct 31, 2007 4:26:32 PM.
Oct 31, 2007 4:26:32 PM com.ibm.tivoli.tuam.logger.JobLogger endJob
INFO: AUCJR0042I Elapsed job time: 3405 milliseconds.
```

Figure 6-3 Running the data transfer job for the Virtual I/O server

Check the output log to ensure that the data transfer is running with no warning or error messages. The processing of the data is shown in Figure 6-4.

```
[root@srv105 bin]# ./startJobRunner.sh LoadVIO.xml
Oct 31, 2007 4:33:02 PM com.ibm.tivoli.tuam.logger.JobLogger startJob
[..]
Oct 31, 2007 4:33:19 PM com.ibm.tivoli.tuam.logger.JobLogger endJob
INFO: AUCJR0033I The LoadVIO job completed successfully at the following
time: Oct 31, 2007 4:33:19 PM.
Oct 31, 2007 4:33:19 PM com.ibm.tivoli.tuam.logger.JobLogger endJob
INFO: AUCJR0042I Elapsed job time: 17225 milliseconds.
```

Figure 6-4 Running the data processing and load for VIOS

Check the output log to ensure that the processing and loading data job is running with no warning or error messages.

6.1.3 AIX Advanced Accounting setup for data collection

Installing the default UNIX collector on AIX will include some scripts for extracting data from AIX Advanced Accounting as well. We run both collectors in parallel in our installation. This section describes how to:

- "Enable AIX Advanced Accounting" on page 165
- "Manually install AIX collector package" on page 166
- "Set up the collector scripts" on page 168
- "Modify the AIX Advanced Accounting job file" on page 170

Enable AIX Advanced Accounting

The setup commands for AIX Advanced Accounting are listed in Example 6-5. We use a five minute aggregation to user level.

Example 6-5 Commands to initialize and start Advanced Accounting on AIX

```
acctctl fadd /var/aacct/aacct0.dat 1
acctctl fadd /var/aacct/aacct1.dat 1
acctctl fadd /var/aacct/aacct2.dat 1
acctctl fadd /var/aacct/aacct3.dat 1
acctctl fadd /var/aacct/aacct4.dat 1
acctctl isystem 5
acctctl iprocess 5
acctctl agproc on
acctctl agke on
acctctl agarm on
```

```
mkitab 'aacct:2:once:/usr/bin/acctctl on >/dev/console 2>&1'

acctctl on
```

The **fadd** subcommand specifies the size of data files, which is 1MB. This is the smallest possible size and is sufficient for our aggregation level. We set the interval record for logging to five minutes and switch on all possible aggregations.

The inittab entry using the **mkitab** command ensures that AIX Advanced Accounting is started on AIX startup. The **acctctl on** command starts the AIX Advanced Accounting immediately. Figure 6-5 shows the output of the **acctctl** command to verify the setup.

```
# acctctl
Advanced Accounting is running.
Email notification is off.
The current email address to be used is not set.
Process Interval Accounting every 5 minutes.
System Interval Accounting every 5 minutes.
System-wide aggregation of process data is on.
System-wide aggregation of third party kernel extension data is on.
System-wide aggregation of ARM transactions is on.
Files: 5 defined, 4 available.
```

Figure 6-5 Check for the state of AIX Advanced Accounting using acctctl command

If you need more detailed information, you must switch off some aggregations. Refer to *AIX 5.3 Advanced Accounting Subsystem,* SC23-4882-03 for further details.

> **Note:** AIX Advanced Accounting switches to the next data file on every reboot. If no free data file is available the accounting will be stopped.

Manually install AIX collector package

In a firewalled environment it might not be possible to use the remote install function described in 5.2.1, “Remote installation of the AIX data collector” on page 129. This section describes how to install the AIX collector package manually.

Begin by transferring the following files from the Tivoli Usage and Accounting Manager server to the target systems /tmp folder:

- /opt/ibm/tuam/collectors/unixlinux/ituam_uc_aix5.tar
- /opt/ibm/tuam/collectors/unixlinux/tuam_unpack_uc_collector

Ensure there are at least 50 MB of free space in the /opt/ibm directory.

On the target system execute the commands in Example 6-6 for installing the standard UNIX data collector.

Example 6-6 Commands for manual installation of AIX collector package

```
chmod 755 /tmp/tuam_unpack_uc_collector
mkdir -p /opt/ibm/tuam/collectors/Unix
cd /opt/ibm/tuam/collectors/Unix
mv /tmp/tuam_unpack_uc_collector .
mv /tmp/ituam_uc_aix5.tar .
./tuam_unpack_uc_collector path=/opt/ibm/tuam/collectors/Unix
```

Verify the results by checking the filesystem and the crontab modification. Figure 6-6 shows our results.

```
root@ohm02:/opt/ibm/tuam/collectors/Unix >du -s *
32      A_README
0       accounting
37792   bin
192     data
640     description
632     etc
160     examples
2040    help
0       history
0       log
1728    scripts

root@ohm02:/opt/ibm/tuam/collectors/Unix >crontab -l | grep -i tuam
# TUAM UNIX/Linux Data Collector scripts
5 1 * * * (/opt/ibm/tuam/collectors/Unix/etc/ituam_uc_nightly 1>
/opt/ibm/tuam/collectors/Unix/log/ituam_uc_nightly.log 2>&1)
3,13,23,33,43,53 * * * * /opt/ibm/tuam/collectors/Unix/etc/check_pacct
5 3 * * *
(/opt/ibm/tuam/collectors/Unix/scripts/enterprise/CS_nightly_consolidation
1> /opt/ibm/tuam/collectors/Unix/log/CS_nightly_consolidation.log 2>&1)
#45 3 * * * (/opt/ibm/tuam/collectors/Unix/scripts/enterprise/CS_send 1>
/opt/ibm/tuam/collectors/Unix/log/CS_send.log 2>&1)
```

Figure 6-6 Check the installation of AIX Advanced Accounting scripts

Remove the installation files:

```
rm /opt/ibm/tuam/collectors/Unix/tuam_unpack_uc_collector
rm /opt/ibm/tuam/collectors/Unix/ituam_uc_aix5.tar
```

Set up the collector scripts

Tivoli Usage and Accounting Manager uses one configuration file and two scripts for data collection from AIX Advanced Accounting.

1. Update the A_config.par configuration file as shown in Example 6-7. The file is located in /opt/ibm/tuam/collectors/Unix/data/A_config.par.

Example 6-7 Updates to the A_config.par file for AIX Advanced Accounting

```
# AIX Advanced Accounting
#  Valid AACCT_TRANS_IDS are 1 4 6 7 8 10 11 16
#
AACCT_TRANS_IDS="1,4,6,7,8,11"
AACCT_ONLY=N
```

   AACCT_ONLY=Y will stop the standard UNIX collector, when using default schedules.

2. Two Tivoli Usage and Accounting Manager scripts have to be scheduled to run at least daily, recognizing the date change for correct processing.

ituam_get_aacct	This script stops the AIX Advanced Accounting, copies the accounting files, and restarts AIX Advanced Accounting.
ituam_format_aacct	This script formats data from AIX Advanced Accounting files.

3. Use the scheduling script shown in Example 6-8 to coordinate execution of the Tivoli Usage and Accounting Manager scripts. Create the script /usr/local/ituam_rb_schedule.sh and make it executable using the **chmod 755 ituam_rb_schedule.sh** command.

Example 6-8 Scheduler script for running AIX Advanced Accounting preprocessing

```
#!/bin/ksh
#
# script for running the adavanced accounting data collection with
exact timing
#
# 2007-11-02 JSiglen initial version
# changes:
#
###################################################################
# set ITUAM Unix collector dir
```

```
ITUAMHOME=/opt/ibm/tuam/collectors/Unix/scripts/aacct
ITUAMGET=${ITUAMHOME}/ituam_get_aacct
ITUAMFORMAT=${ITUAMHOME}/ituam_format_aacct
#
# get current date at start
START=$(date +%Y%m%d)
# call ITUAMGET
${ITUAMHOME}/ituam_get_aacct
# wait for 10 minutes (600 sec)
sleep 600
# check if date has changed
if [ ${START} -eq $(date +%Y%m%d) ]; then
        echo same date - no preprocessing needed
else
        # run format script with preday date
        ${ITUAMFORMAT} ${START}
fi
```

4. Comment out the existing Tivoli Usage and Accounting Manager crontab entries and create new ones as listed in Example 6-9.

Example 6-9 Crontab entries for data extraction from AIX Advanced Accounting

```
# ITUAM UNIX/Linux Data Collector scripts
#
#5 1 * * * (/opt/ibm/tuam/collectors/Unix/etc/ituam_uc_nightly 1>
/opt/ibm/tuam/collectors/Unix/log/ituam_u
c_nightly.log 2>&1)
#3,13,23,33,43,53 * * * * /opt/ibm/tuam/collectors/Unix/etc/check_pacct
#5 3 * * *
(/opt/ibm/tuam/collectors/Unix/scripts/enterprise/CS_nightly_consolidat
ion 1> /opt/ibm/tuam/coll
ectors/Unix/log/CS_nightly_consolidation.log 2>&1)
# AIXAA collector schedules
59 * * * * /usr/local/ituam_rb_schedule 1>
/opt/ibm/tuam/collectors/Unix/log/ituam_rb_schedule.log 2>&1
```

We run ituam_rb_schedule hourly to ensure having enough space for our accounting data files. It is also useful for hourly aggregation later on. Once the date changes, the script runs ituam_format_aacct, using the same date as the file contents.

5. The A_config.par contains additional parameters for cleanup, which must be set based on the available disk space. Using the settings in Example 6-10 will provide 7 days of source accounting data and 45 days of preprocessed data.

Example 6-10 Setting the cleanup variables for the AIX data collector

```
CLEANUP_HISTORY=Y
CLEANUP_AGE=+7

CLEANUP_ACCT=Y
CLEANUP_ACCT_AGE=+45
```

> **Note:** The history files are allocated hourly, each with a size of 1 MB. For 7 days of data, you would need around 168 MB (24 x 7).

Modify the AIX Advanced Accounting job file

The processing of the AIX advanced accounting is shown in Figure 6-7.

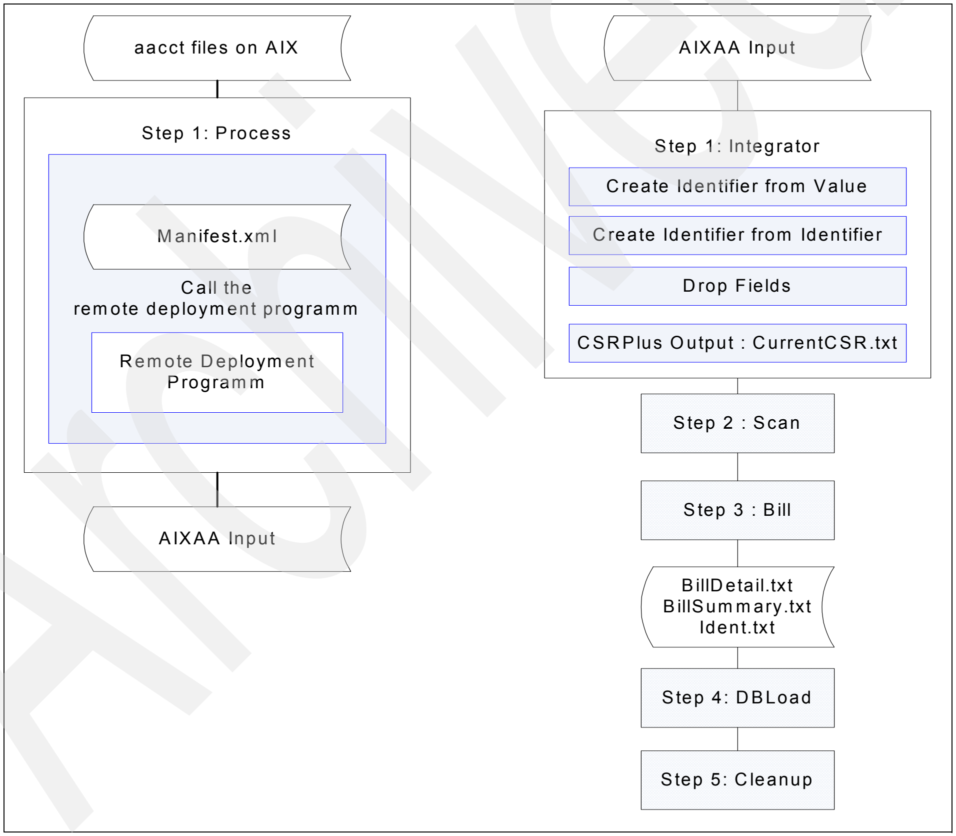


Figure 6-7 AIX Advanced Accounting process flow

The processing consists of transferring and loading AIX advanced accounting data files. We use the same two-job structure as for the VIOS data.

Transferring AIX Advanced Accounting data

Copy the sample files of the VIOS data transfer, SampleSecureGetVIOS* from "Prepare for transferring VIOS data" on page 162. We called the new job file TransferAIXAA.xml. Update the parameters in the TransferAIXAA.xml as shown in Example 6-11.

Example 6-11 TransferAIXAA.xml parameter updates

```
<Parameters>
   <Parameter Host="lpar04"/>
   <Parameter UserId="root"/>
   <Parameter Password="******"/>
   <Parameter Manifest="TransferAIXManifest.xml"/>
   <Parameter
   RPDParameters="client_CS_path=/opt/ibm/tuam/collectors/Unix/CS_input
   _source;CollectorLogs_dir=%CollectorLogs%/AIXAA;LogDate=%LogDate_End
   %;client_name=lpar04;"/>
   <Parameter Verbose="true"/>
   <Parameter SourcePath="%HomePath%/collectors/unixlinux/"/>
</Parameters>
```

The meanings of the parameters are:

Host	DNS name or IP of the VIOS server
Manifest	Name of the RPD action definition
client_CS_path	Path of the aacct files on the VIOS server
CollectorLogs_dir	Base directory for VIOS logs on the Tivoli Usage and Accounting Manager server
client_name	Name for the subdirectory "Feed" on the Tivoli Usage and Accounting Manager server
SourcePath	Path where the manifest file is located

Run the transfer job using the rpd program.

> **Important:** The user must be allowed to use **`sftp`** on the designated client.

The TransferAIXAAManifest.xml has to be updated. Example 6-12 shows the modification in the manifest file for collecting aacct_11 file. Remove all comments in <action> sections and add a section for aacct_11 file (VIOS data).

Example 6-12 Changes needed in TransferAIXAAManifest.xml

```
<Action name="step_AACCT_11_%client_name%"
   displayMessage="Getting nightly AACCT_11 file for %client_name%"
   actionType="FileGet">
   <Parameters>
```

```
      <Parameter name="localpath"
         value="%CollectorLogs_dir%/%client_name%" />
      <Parameter name="remotefilename"
         value="%client_CS_path%/aacct11_%LogDate%.txt" />
   </Parameters>
</Action>
```

After running the job, get the data files in the collectors directory of the Tivoli Usage and Accounting Manager server. See Example 6-13.

Example 6-13 Running the data transfer from AIX for Advanced Accounting data

```
[root@srv105]# cd /opt/ibm/tuam/logs/collectors/AIXAA
[root@srv105]# /opt/ibm/tuam/bin/startJobRunner.sh TransferAIXAA.xml
Oct 30, 2007 5:24:10 PM com.ibm.tivoli.tuam.logger.JobLogger startJob
   .  .  .
INFO: AUCJR0042I Elapsed job time: 1927 milliseconds.
[root@srv105]# ls -lt lpar04
-rw-r--r--  1 root root  124013 Oct 30 17:24 aacct11_20071029.txt
-rw-r--r--  1 root root   98030 Oct 30 17:24 aacct4_20071029.txt
-rw-r--r--  1 root root  735627 Oct 30 17:24 aacct6_20071029.txt
-rw-r--r--  1 root root   87343 Oct 30 17:24 aacct7_20071029.txt
-rw-r--r--  1 root root  233427 Oct 30 17:24 aacct8_20071029.txt
-rw-r--r--  1 root root   41206 Oct 30 17:24 aacct1_20071029.txt
-rw-r--r--  1 root root  118925 Oct 30 11:05 aacct11_20071028.txt
-rw-r--r--  1 root root  705743 Oct 30 11:05 aacct6_20071028.txt
-rw-r--r--  1 root root   83800 Oct 30 11:05 aacct7_20071028.txt
-rw-r--r--  1 root root  223849 Oct 30 11:05 aacct8_20071028.txt
-rw-r--r--  1 root root   38711 Oct 30 11:05 aacct1_20071028.txt
-rw-r--r--  1 root root   94081 Oct 30 11:05 aacct4_20071028.txt
```

Processing and loading AIX Advanced Accounting data

Copy the SampleAIXAA.xml into LoadAIXAA.xml. The path for the input data has to be adjusted as shown in Example 6-14.

Example 6-14 LoadAIXAA.xml Input section

```
<Input name="AIXAAInput" active="true">
 <Files>
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct1_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct4_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct6_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct7_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct8_%LogDate_End%.txt" />
  <File name="%CollectorLogs%/AIXAA/lpar04/aacct11_%LogDate_End%.txt" />
```

```
  <File name="%ProcessFolder%/exception.txt" type="exception" />
 </Files>
</Input>
```

The integrator step formats the Account_code identifier. Perform the following procedures:

- Create an Account_Code_TMP using a lookup table from the SYSTEM_ID field as shown in Example 6-15.

Example 6-15 Account code lookup

```
<Stage name="CreateIdentifierFromTable" active="true">
 <Identifiers>
  <Identifier name="Account_Code_TMP">
   <FromIdentifiers>
    <FromIdentifier name="SYSTEM_ID" offset="1" length="10" />
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Files>
  <File name="/opt/ibm/tuam/processes/Accttabl.txt" type="table" />
  <File name="Exception.txt" type="exception" format="CSROutput" />
 </Files>
 <Parameters>
  <Parameter exceptionProcess="true" />
  <Parameter sort="true" />
  <Parameter upperCase="false" />
  <Parameter writeNoMatch="false" />
  <Parameter modifyIfExists="true" />
 </Parameters>
</Stage>
```

- Append the SYSTEM_ID (hostname) to the temporary account code, as shown in Example 6-16.

Example 6-16 Appending SYSTEM_ID

```
<Stage name="CreateIdentifierFromIdentifiers" active="true">
 <Identifiers>
  <Identifier name="Account_Code">
   <FromIdentifiers>
    <FromIdentifier name="Account_Code_TMP" offset="1" length="40" />
    <FromIdentifier name="SYSTEM_ID" offset="1" length="20" />
   </FromIdentifiers>
  </Identifier>
```

```
 </Identifiers>
 <Parameters>
  <Parameter keepLength="true" />
  <Parameter modifyIfExists="true" />
 </Parameters>
</Stage>
```

- Remove the temporary account code field, Account_Code_TMP.
- Set the SYSTEM_ID identifier to the serial number (Example 6-17) for normalization of the CPU based on the hardware used.

Example 6-17 Formatting the serial number for normalization of AIXAA CPU values

```
<Stage name="CreateIdentifierFromRegEx" active="true" trace="true">
 <Identifiers>
  <Identifier name="SYSTEM_ID">
   <FromIdentifiers>
    <FromIdentifier name="SysId" regEx="IBM,(.{9}).*" value="$1" />
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter modifyIfExists="true" />
 </Parameters>
</Stage>
```

Figure 6-18 shows the updated path for the CSR output file.

Example 6-18 Updating CSR output path for AIXAA data

```
<Files>
 <File name="%ProcessFolder%/lpar04/%LogDate_End%.txt" />
</Files>
```

We set up a simple accounting table for assigning accounts based on the first letter of hostname to one of our accounts. Account mapping in the AcctTbl.txt file is shown in Example 6-19. All hostnames starting with `k` will be assigned to `AIX 0TEST` and those starting with `l` through `n` will be assigned to `AIX 1TEST`.

Example 6-19 Account mapping table

```
k,,AIX 0TEST
l,n,AIX 1TEST
r,,LIN 0TEST
s,t,LIN 1TEST
```

CPU normalization

All CPU values are normalized, based on the SYSTEM_ID identifier, when the controlCard statement for NORMALIZE CPU VALUES is added as shown in Example 6-20.

Example 6-20 AIX Advanced Accounting billing and normalization

```
<Step id="Process" description="Billing" type="Process"
programName="Bill" programType="java" active="true">
 <Bill>
  <Parameters>
   <Parameter inputFile="CurrentCSR.txt" />
   <Parameter controlCard="NORMALIZE CPU VALUES" />
  </Parameters>
 </Bill>
</Step>
```

The normalization table must match the SYSTEM_ID and the Rate Code must be set to CPU value. We have to enter the serial number into the CPU Normalization table using the Integrated Solutions Console (ISC). Figure 6-8 on page 176 shows the ISC dialog accessed by the menu selection **Usage and Accounting Manager** → **System Maintenance** → **CPU Normalization** → **New.**

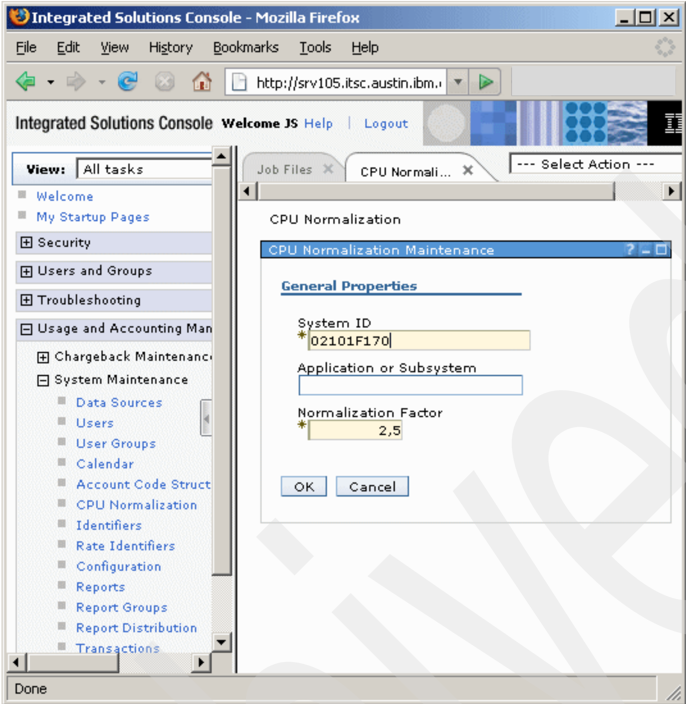


Figure 6-8 Adding a CPU Normalization value

Check for marked CPU value in the Rate definitions (Figure 6-9 on page 177)

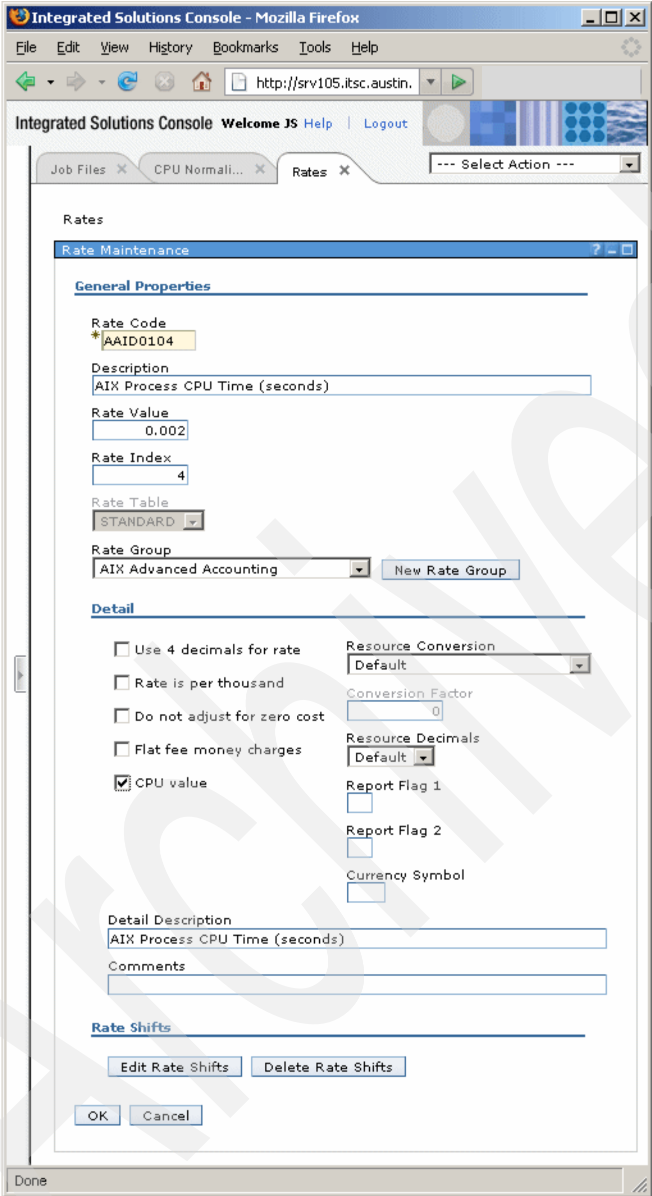


Figure 6-9 Rate definition for CPU values which need normalization

Running the job, we process and load the data in the Tivoli Usage and Accounting Manager DB. The job ends successfully and there are no warnings or errors in the log file.

6.1.4 Reporting on AIX Advanced Accounting

To view reports on usage, point your browser to `http://srv177/tuam`.

Figure 6-10 shows the data transfer for the VIOS server (serv) and our AIX VIOS client (AIX). The VIOS server is providing disk space for other partitions also; therefore we can see more I/O at “serv.”

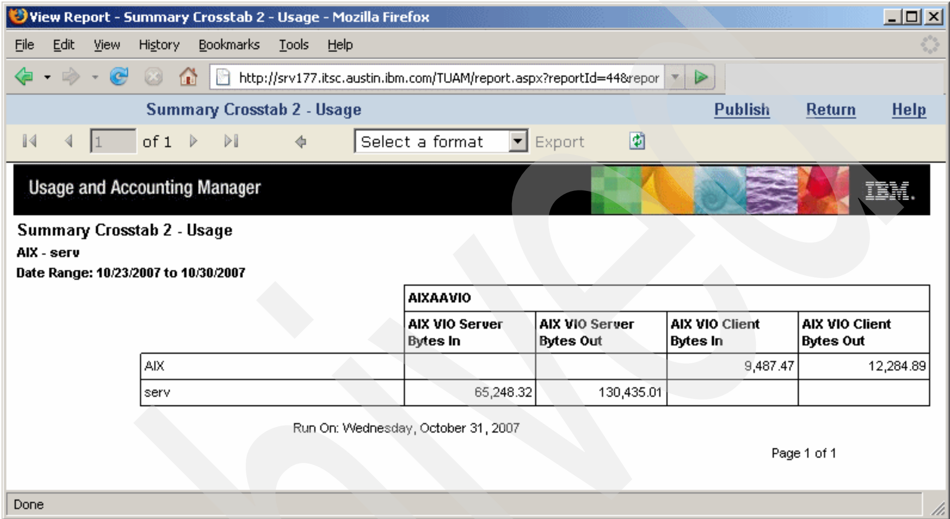
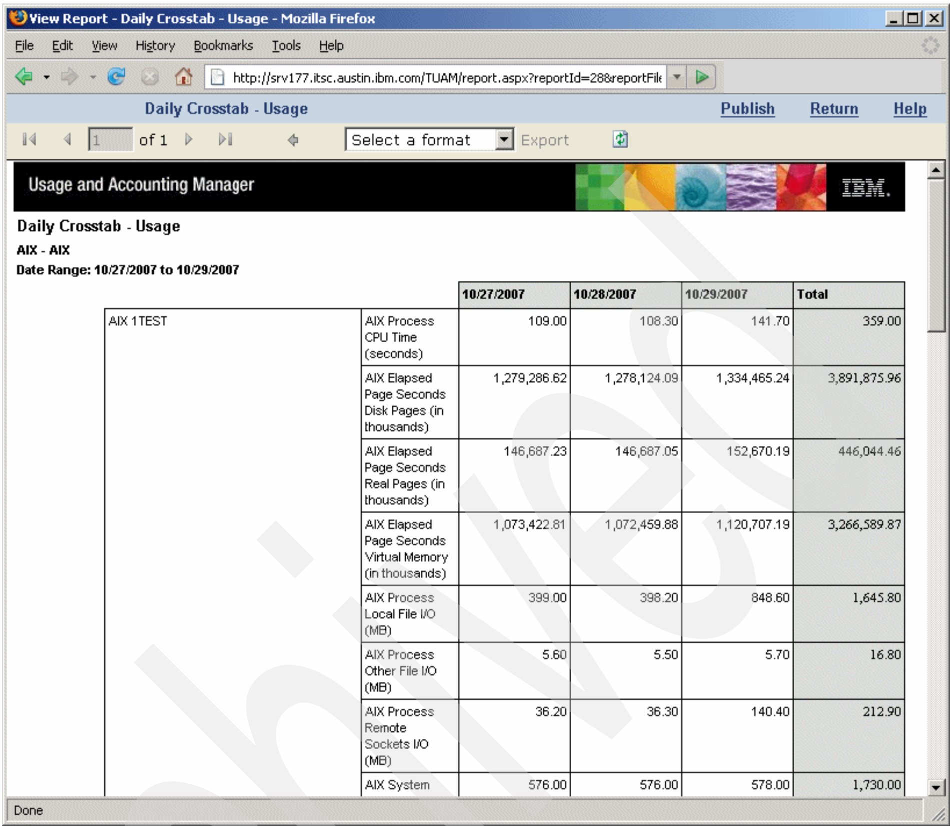


Figure 6-10 VIOS reporting

An extract of the daily crosstab for AIX is shown in Figure 6-11 on page 179



		10/27/2007	10/28/2007	10/29/2007	Total
AIX 1TEST	AIX Process CPU Time (seconds)	109.00	108.30	141.70	359.00
	AIX Elapsed Page Seconds Disk Pages (in thousands)	1,279,286.62	1,278,124.09	1,334,465.24	3,891,875.96
	AIX Elapsed Page Seconds Real Pages (in thousands)	146,687.23	146,687.05	152,670.19	446,044.46
	AIX Elapsed Page Seconds Virtual Memory (in thousands)	1,073,422.81	1,072,459.88	1,120,707.19	3,266,589.87
	AIX Process Local File I/O (MB)	399.00	398.20	848.60	1,645.80
	AIX Process Other File I/O (MB)	5.60	5.50	5.70	16.80
	AIX Process Remote Sockets I/O (MB)	36.20	36.30	140.40	212.90
	AIX System	576.00	576.00	578.00	1,730.00

Figure 6-11 AIXAA daily detail report extract

6.1.5 AIX Advanced Accounting burst calculation

Using the System p hardware, you can reserve some CPU capacity for a partition and allow additional use of unused CPU cycles. This facility is called the *burst CPU time*.

In “Sample job for AIX CPU burst calculation” on page 344, we list a job for processing the CPU burst, splitting the real CPU usage from the reserved capacity, and charging the CPU burst for billing with different rates.

Figure 6-12 on page 180 shows the functional overview for the burst calculation.

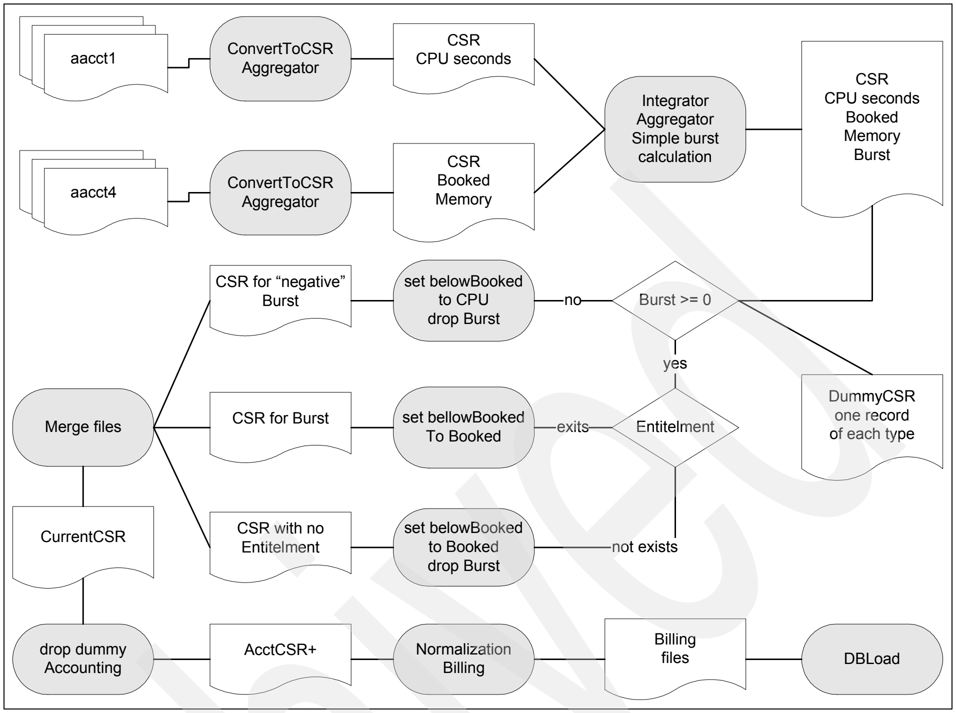


Figure 6-12 Overview on AIX Advanced Accounting burst calculation

The steps used for calculating the booked and burst values follow.

1. The aacct1 file containing the process data and the aacct4 file with all environmental settings are aggregated at the hour level and saved in CSR format. The entitlement in aacct4 is converted into a BOOKED value based on CPU seconds:

   ```
   BOOKED = (average Entitelment) * 3600 / 100
   ```

2. The Integrator can now merge the two files and subtract the BOOKED from the measured CPU seconds (in aacct1) to calculate a BURST value.

   ```
   BURST = (CPU seconds) - BOOKED
   ```

3. Three different Integrator steps handle the different situations.

   To avoid warnings on empty files, we create records from all three types in the DummyCSR.txt, which are included into the processing.

BURST is negative	There was no BURST, so the Resource can be dismissed and the belowBooked is set to the CPU seconds measured.

Entitlement exits — The BURST value is OK and belowBooked is set to BOOKED.

Entitlement does not exist — No BURST calculation can be done because the missing entitlement is taken as zero entitlement. The BURST value will be discarded for not being charged. The belowBooked will be set to BOOKED (which will be zero anyway). A zero BURST (full loaded, capped LPAR) will be handled within this step like no entitlement.

4. All files are scanned into one file for further processing.
5. The records from DummyCSR are removed and the account code conversion is done.
6. Normalization and Billing is performed as it was for the standard AIX described previously.
7. The data is loaded into the Tivoli Usage and Accounting Manager database.

Figure 6-13 on page 182 shows a sample output for the usage data of AIX systems.

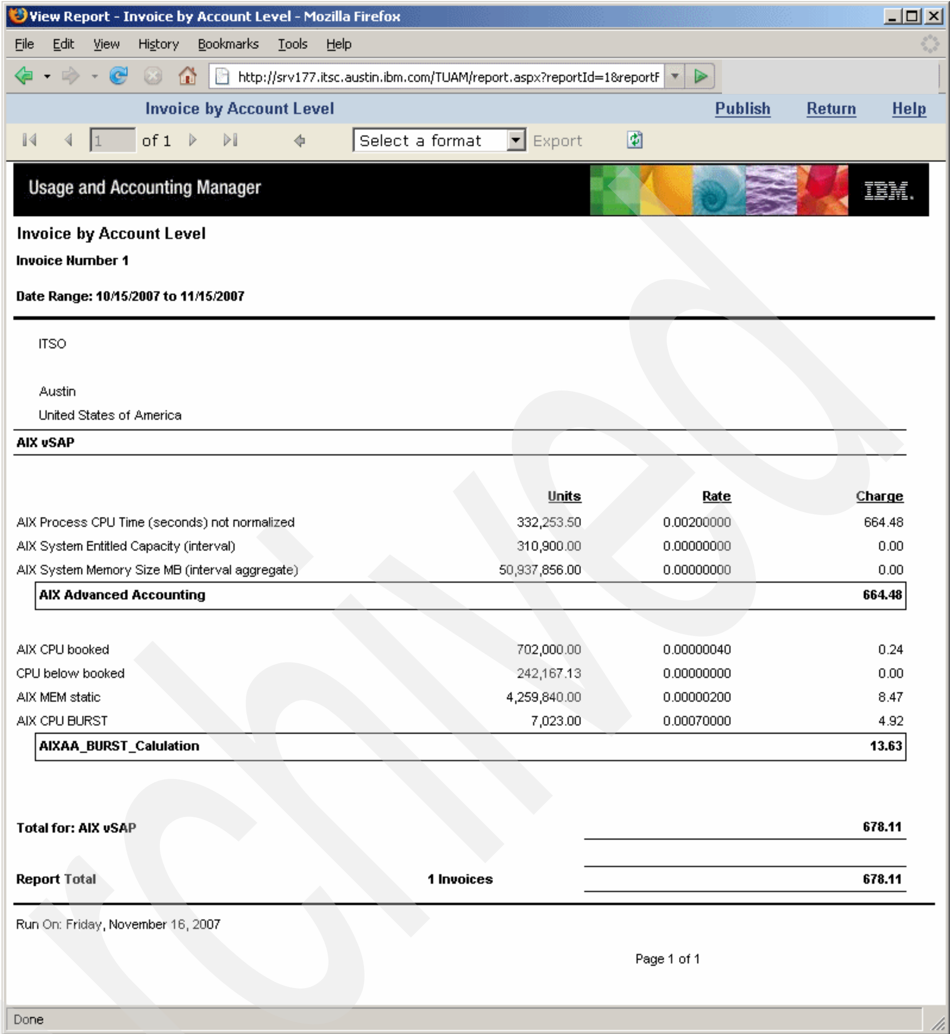


Figure 6-13 Sample report on AIX Advanced Accounting burst data

The last four items are from the Burst calculation; the rest is left from previous collections in our lab environment.

6.2 Virtualization with VMware

The VMware virtualization discussion is divided into:

- 6.2.1, "Architecture for VMware scenario" on page 183

- 6.2.2, “Prerequisites on the VMware VirtualCenter Server” on page 184
- 6.2.3, “Tivoli Usage and Accounting Manager server configuration” on page 185
- 6.2.4, “Building the VMware collection job file” on page 189
- 6.2.5, “Reporting on VMware Accounting” on page 198

6.2.1 Architecture for VMware scenario

Figure 6-14 shows the scenario for VMware VirtualCenter collector.

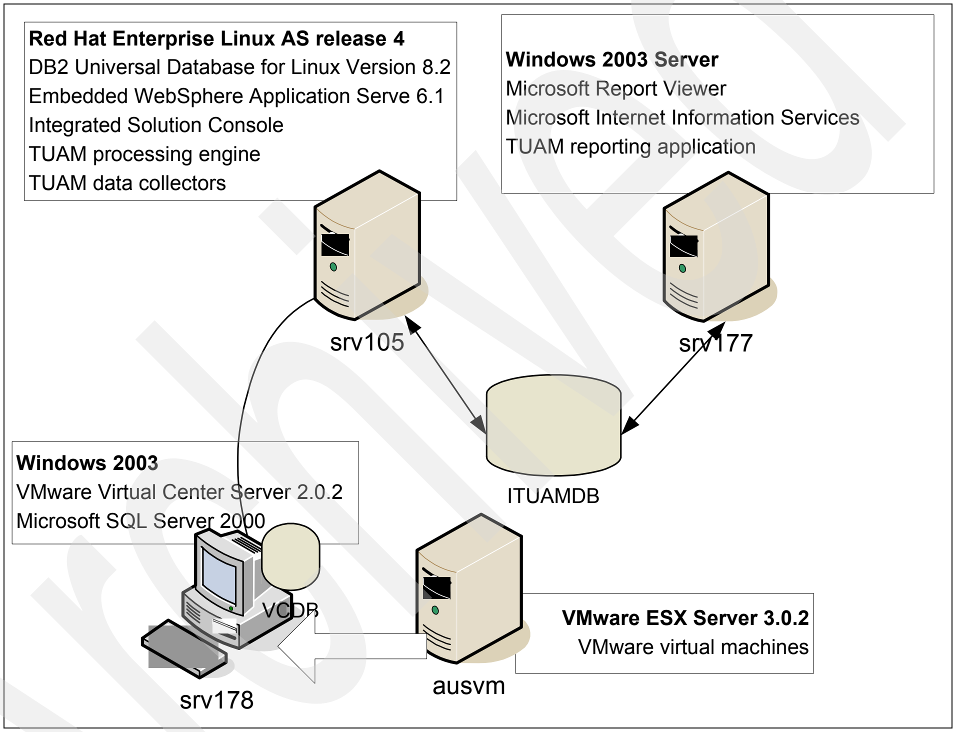


Figure 6-14 Virtualization scenario with VMware VirtualCenter Server

VMware Virtual Center Server collects all statistical data from the connected VMware ESX Servers. Tivoli Usage and Accounting Manager accesses this data by using the VMware Software Development Kit (SDK). The SDK is a set of Java libraries that perform Web Service calls to the Virtual Center Server.

Data in the VirtualCenter database (VCDB) is aggregated by default. The SDK interface that is used by Tivoli Usage and Accounting Manager can only provide the aggregation levels provided by VCDB.

6.2.2 Prerequisites on the VMware VirtualCenter Server

Some setup steps must be performed on the VirtualCenter Server before data collection can start.

- VirtualCenter Server 2.0 or later is recommended.

Note: When VirtualCenter Server 1.4 is used, it must have been installed using the Custom Install option to include Web services for the SDK.

- Configuration of the Web service on the VMware Virtual Center Server for use of the VMware Software Development Kit (SDK) must have been done according to *Installation and Upgrade Guide for ESX 3.0.1 and VirtualCenter 2.0.1*.

Note: VirtualCenter Server uses HTTPS by default. If you prefer to use the non secure HTTP you have to update the C:\Documents and Settings\All Users\Application Data\VMware\VMware VirtualCenter\vpxd.cfg and restart the VMware services.

- Windows services of VMware Virtual Center Server must be set to automatic start and must be started, as shown in Figure 6-15.

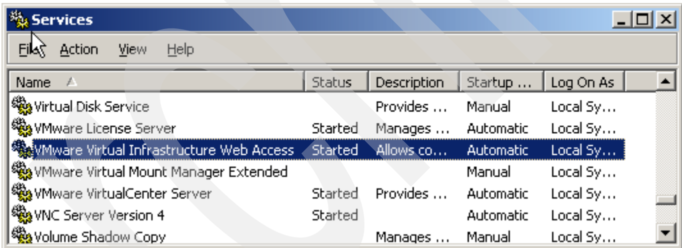


Figure 6-15 The Windows services on the VirtualCenter Server

- The Statistics Collection Level has to be set to Level 3 in the VirtualCenter Server menu **Administration** → **VirtualCenter Management Server Configuration...** → **Statistics** as shown in Figure 6-16 on page 185.

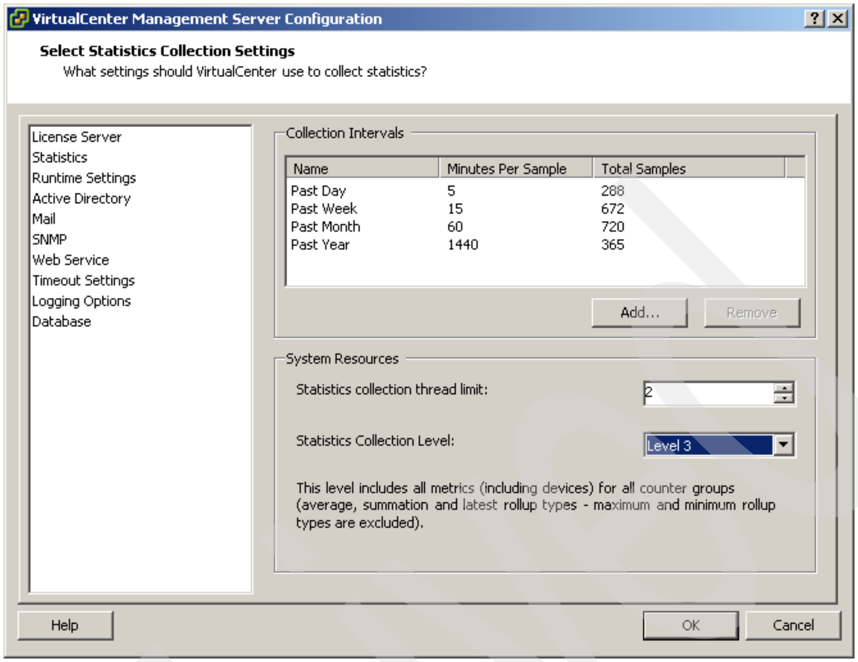


Figure 6-16 VirtualCenter collection settings

- Your ESX servers must be defined as part of the VirtualCenter Server data center for collecting the usage data.

6.2.3 Tivoli Usage and Accounting Manager server configuration

The VirtualCenter Server is defined as a Data Source of type Collector - Web Service within the Tivoli Usage and Accounting Manager configuration. This section provides the details about the following high-level configuration steps:

- “Getting the SDK package” on page 185
- “Applying the SDK jar file” on page 186
- “Loading the SSH key” on page 187
- “Set up Web Service for VMware data collector” on page 187

Getting the SDK package

Tivoli Usage and Accounting Manager uses the VMware Infrastructure SDK package. The SDK jar file must be downloaded to the Tivoli Usage and Accounting Manager application server. Perform the following steps:

1. Browse to the VMware Web site:

   `http://www.vmware.com/download/sdk/`

2. Click the download link for VMware Infrastructure SDK Packages.
3. Log in with your user ID and password; you may have to register for login first.
4. Accept the license agreement.
5. Click the VMware Infrastructure SDK 2.0.1 download link ZIP image and save the file.
6. Extract the vim.jar into the lib subdirectory of the Tivoli Usage and Accounting Manager server as shown in Figure 6-17 and copy the file to the ewas library path.

```
# cd /opt/ibm/tuam/lib
# unzip -j /tmp/vi-sdk-2.0.1-32042.zip SDK/samples_2_0/Axis/java/vim.jar
Archive:  /ti7b55/noback/vmware/vi-sdk-2.0.1-32042.zip
  inflating: vim.jar
# cp -p /opt/ibm/tuam/lib/vim.jar
/opt/ibm/tuam/ewas/systemApps/isclite.ear/aucConsole.war/WEB-INF/lib
```

Figure 6-17 Extracting VMware SDK

7. Stop and restart the application server for activation of the new library:

```
/opt/ibm/ibm/tuam/bin/stopServer.sh
/opt/ibm/ibm/tuam/bin/startServer.sh
```

Applying the SDK jar file

Tivoli Usage and Accounting Manager runs a job from the command line startJobRunner.sh or from the Integrated Solution Console.

To use the command line **startJobRunner.sh**, we must add the classpath as shown in Example 6-21 to include the vim.jar file. The file is normally located in /opt/ibm/tuam/bin/startJobRunner.sh.

Example 6-21 Add the vim.jar in the startJobRunner.sh command script

```
# **** need to add vim.jar if you want to use VMWare collector ******
JR_CLASSPATH=$JR_CLASSPATH:$TUAM_LIB/vim.jar
```

For running the VMware collection using ISC, we already copied the file into the embedded WebSphere library path.

Note: If the latest SDK version does not work, you can try the previous version instead. VMware SDK may have changed.

Loading the SSH key

The ssh key from the VMware Virtual Center HTTPS document can be found in C:\Documents and Settings\All Users\Application Data\VMware\VMware Virtual Center\SSL\rui.crt. Transfer the rui.crt to the Tivoli Usage and Accounting Manager application server.

Figure 6-18 shows how to load the VMware SSH keys in a keystore on the Tivoli Usage and Accounting Manager server.

```
# /opt/ibm/tuam/_jvm/bin/keytool -import -file /tmp/rui.crt -alias
srv178 -keystore /etc/vmware-sdk/vmware.keystore
Enter keystore password:  ********
Owner: EMAILADDRESS=support@vmware.com, CN=VMware, OU="VMware,
Inc.", O="VMware, Inc.", L=CA, ST=CA, C=US
Issuer: EMAILADDRESS=support@vmware.com, CN=VMware, OU="VMware,
Inc.", O="VMware, Inc.", L=CA, ST=CA, C=US
Serial number: fe9bfd2f651768d6
Valid from: 10/4/07 6:07 PM until: 10/3/09 6:07 PM
Certificate fingerprints:
         MD5:  50:34:49:95:FD:67:1C:8A:D5:6D:DF:35:B2:67:EE:28
         SHA1:
93:1C:43:6F:93:62:B0:CE:B0:C4:9B:2A:DB:C8:58:9B:C4:29:F8:71
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

Figure 6-18 Loading the ssh keys into a keystore for using HTTPS connection

Set up Web Service for VMware data collector

In the ISC, select **Usage and Accounting Manager** → **System Maintenance** → **Data Sources** → **Collector - Web Service** to view the defined Collector - Web Services as shown in Figure 6-19 on page 188.

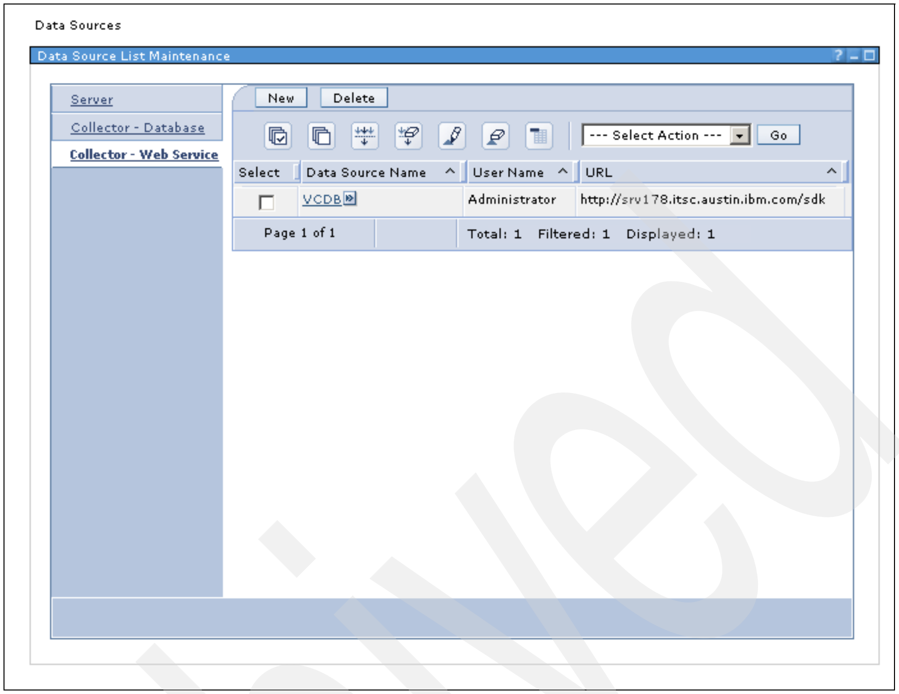


Figure 6-19 List of defined Collector-Web Services in the ISC

Click **New** to add a data source as shown in and Figure 6-20.

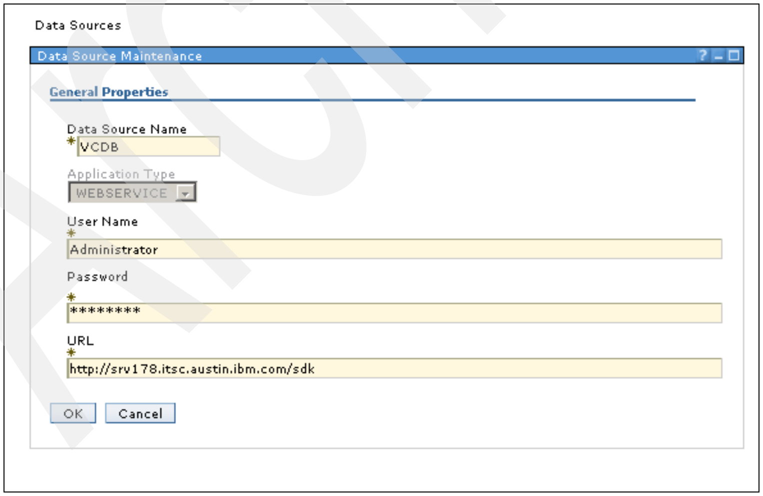


Figure 6-20 Set up Collector-Web Service data source

You may need to provide some port information in the URL string depending on your installation (such as `http://server:port/sdk`).

Restriction: The Data Source Name must not exceed 8 characters, otherwise the VMware job would not be able to find the data source.

6.2.4 Building the VMware collection job file

This section describes how to build a job file to transfer and load the accounting data from VirtualCenter server.

Overview of VMware data processing

The process overview for VMware as shown in Figure 6-21 is doing the data transfer and load all in one job. To identify the VMware guests, and for CPU Normalization, we have to create new identifiers.

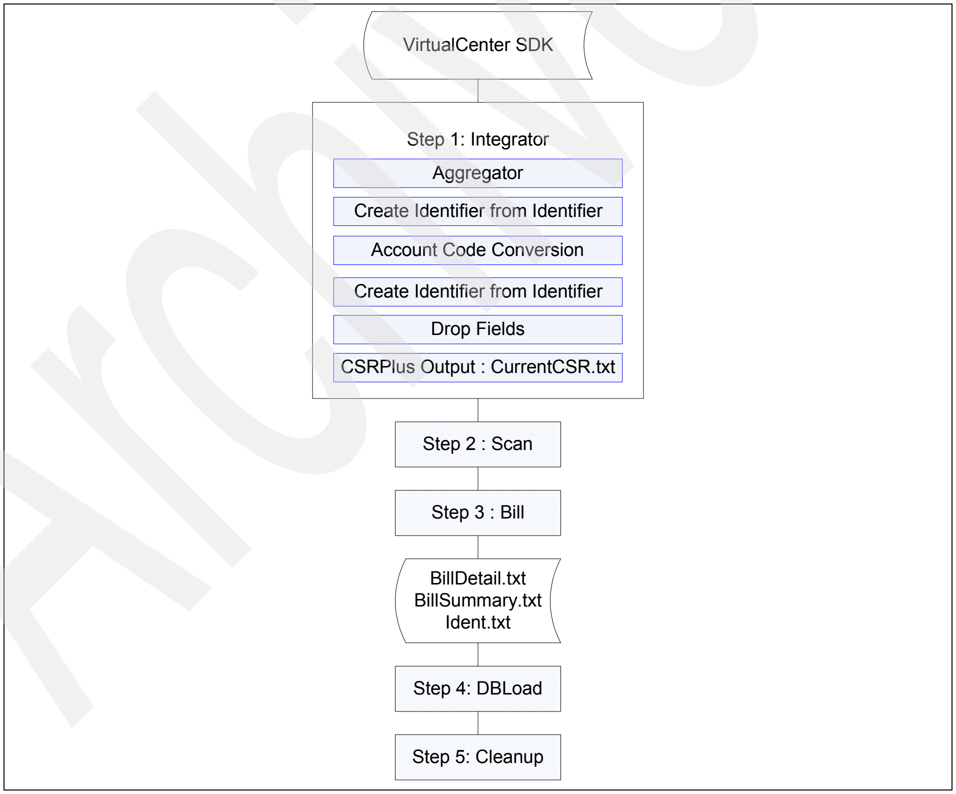


Figure 6-21 VMware processing overview

Preparation for the VMware job file

Copy the sample job file SampleVMWare.xml into LoadVMware.xml. Set the Integrator step to `active="true"`. The connection parameters have to be modified as shown in Example 6-22.

Example 6-22 he updated VMware Collector section

```
<Input name="CollectorInput" active="true">
   <Collector name="VMWARE">
      <WebService dataSourceName="VCDB"
      certificateStore="/etc/vmware-sdk/vmware.keystore" />
      <ManagedEntity type="VIRTUALMACHINE"/>
      <Interval id="3600"/>
      <Host dnsName=""/>
   </Collector>
      <Parameters>
         <Parameter name="Feed" value="ITSC_VC" DataType="String"/>
         <Parameter name="logdate" value="%LogDate_End%"
         DataType="DateTime" format="yyyyMMdd"/>
      </Parameters>
   <Files>
      <File name="%ProcessFolder%/Exception_%LogDate_End%.txt"
      type="exception" />
   </Files>
</Input>
```

The parameters are:

dataSourceName	Refers to the Web-Service defined previously, in “Set up Web Service for VMware data collector.”
certificateStore	SSH key as created previously, in “Loading the SSH key.”
Managed Entity	We use `VIRTUALMACHINE`, which is selecting data from VMware guests. The entry `HOST` collects base data from the ESX servers only.
Interval id	This has to match one aggregation level set up previously, in “Prerequisites on the VMware VirtualCenter Server” on page 184. The interval id is just selecting the aggregation level to be summarized by the SDK, so the difference for one day will be insignificant when changing this.
Host dnsName	Used to restrict data collection for one ESX host.
Feed	The subdirectory for the data collected from this VirtualCenter server.

Note: Parameter certificateStore is required even when using HTTP.

- Set the aggregator stage to have the argument `active="true"`.

> **Important:** Correct the `<Identifier name="VmName" />` (small `m`).

The account code conversion for the VMware system is as follows:

- Create a new identifier called HOST_VM_ID from the HostName and VmName identifiers (Example 6-23).

Example 6-23 Account Code Conversion for VMware data

```
<!-- create unique identifier for VMware guest system -->
<Stage name="CreateIdentifierFromRegEx" active="true" trace="true" >
 <Identifiers>
  <Identifier name="HOST_VM_ID">
   <FromIdentifiers>
    <FromIdentifier name="HostName" regEx="host-(.*)" value="$1"/>
    <FromIdentifier name="VmName" regEx="vm(-.*)" value="$1"/>
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
</Stage>
```

- Define the temporary account code from a table based on the HOST_VM_ID identifier (Example 6-24).

Example 6-24 Account Code Conversion for VMware data

```
<!-- get account code fom table based on SYSTEM_ID (hostname) -->
<Stage name="CreateIdentifierFromTable" active="true">
 <Identifiers>
  <Identifier name="Account_Code_TMP">
   <FromIdentifiers>
    <FromIdentifier name="HOST_VM_ID" offset="1" length="10" />
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Files>
  <File name="/opt/ibm/tuam/processes/Accttabl.txt" type="table" />
  <File name="Exception.txt" type="exception" format="CSROutput" />
 </Files>
 <Parameters>
  <Parameter exceptionProcess="true" />
  <Parameter sort="true" />
  <Parameter upperCase="false" />
  <Parameter writeNoMatch="false" />
```

```
  <Parameter modifyIfExists="true" />
 </Parameters>
</Stage>
```

- Append the HOST_VM_ID to the Account_Code identifier (Example 6-25).

Example 6-25 Account Code Conversion for VMware data

```
<!-- add HOST_VM_ID as last part to the account code -->
<Stage name="CreateIdentifierFromIdentifiers" active="true">
 <Identifiers>
  <Identifier name="Account_Code">
   <FromIdentifiers>
    <FromIdentifier name="Account_Code_TMP" offset="1" length="40" />
    <FromIdentifier name="HOST_VM_ID" offset="1" length="10" />
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter modifyIfExists="true" />
  <Parameter keepLength="true" />
 </Parameters>
</Stage>
```

- Add the indentifier SYSTEM_ID for normalization in billing (Example 6-26).

Example 6-26 Create identifier SYSTEM_ID for VMware accounting

```
<Stage name="CreateIdentifierFromIdentifiers" active="true"
trace="false" stopOnStageFailure="true">
 <Identifiers>
  <Identifier name="SYSTEM_ID">
   <FromIdentifiers>
    <FromIdentifier name="DnsName" offset="1" length="30" />
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter keepLength="false" />
  <Parameter modifyIfExists="true" />
 </Parameters>
</Stage>
```

- Update the CSR output path as in Example 6-27.

Example 6-27 Update the CSR output file

```
<Stage name="CSRPlusOutput" active="true">
 <Files>
  <File name="%ProcessFolder%/ITSC_VC/%LogDate_End%.txt" />
 </Files>
</Stage>
```

- Enable the normalization by adding a controlCard statement (Example 6-28).

Example 6-28 Add controlCard to the billing step

```
<Parameters>
 <Parameter inputFile="CurrentCSR.txt" />
 <Parameter controlCard="NORMALIZE CPU VALUES" />
</Parameters>
```

Additional setup needed for VMware job file

We set up a simple accounting table for assigning account codes (Example 6-29).

Example 6-29 Accttab.txt content for VMware account code conversion

```
19-31,19-31,WIN 0ESX
19-33,19-33,WIN 1ESX
19-37,19-37,WIN 2ESX
19-40,19-43,WIN 4ESX
19-63,19-63,WIN 9ESX
```

For any VMware guest we are assigning the WIN Account and a dedicated sub account. All hosts from vm-40 to vm-43 are assigned to the same account “WIN 4ESX.”

> **Tip:** The VirtualCenter console names of VMware guests in the GUI and their relation to VMID, HOSTID can be looked up in the VirtualCenter database view dbo.VPXV_VMS.

Enter the DnsName of the ESX server into the CPU Normalization table using ISC as shown in Figure 6-22. On ISC select **Usage and Accounting Manager** → **System Maintenance** → **CPU Normalization** → **New**.

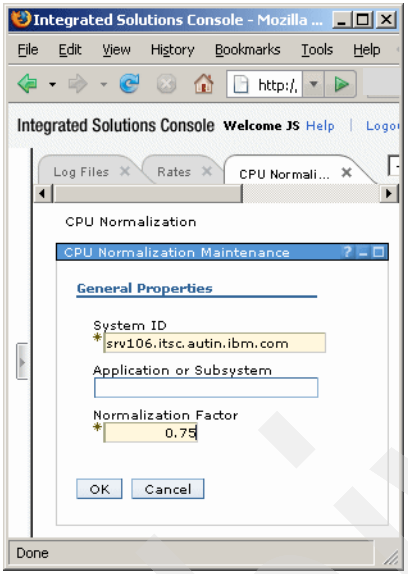


Figure 6-22 Adding a CPU Normalization value for VMware ESX server

Check for marked CPU value in the Rate definitions (Figure 6-23 on page 195)

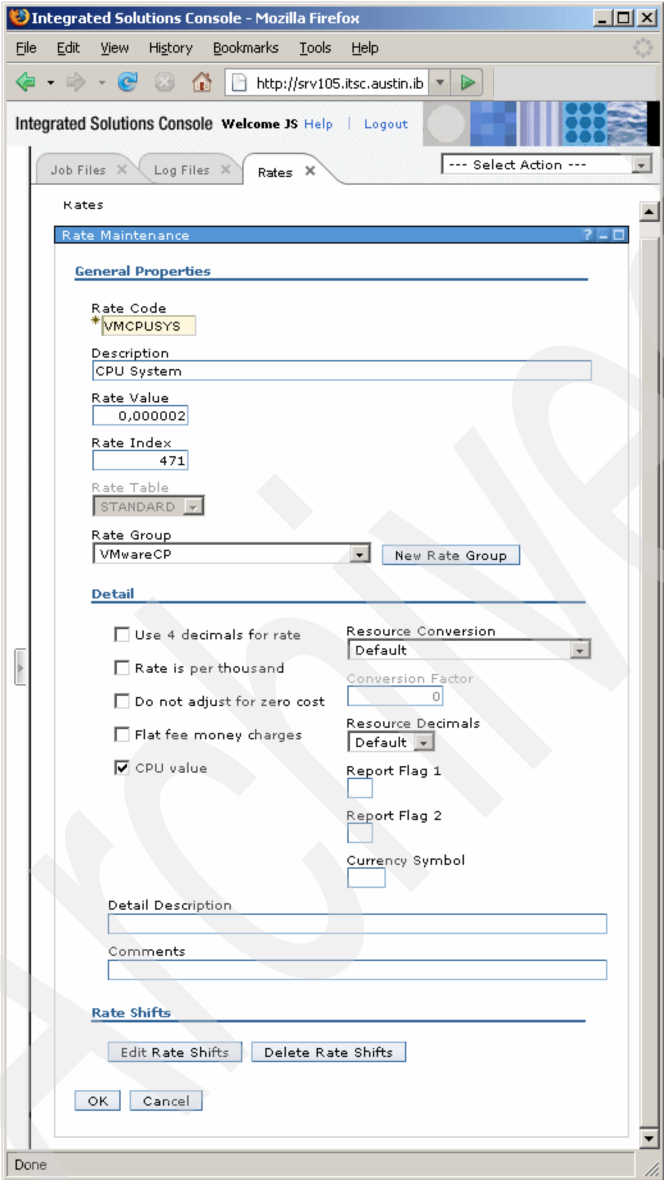


Figure 6-23 Rate definition for CPU values, which needs normalization

Executing the new VMware job file

To run the job using the Integrated Solutions Console, select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files** → **LoadVMware.xml** and click the **Run Job** button (Figure 6-24).

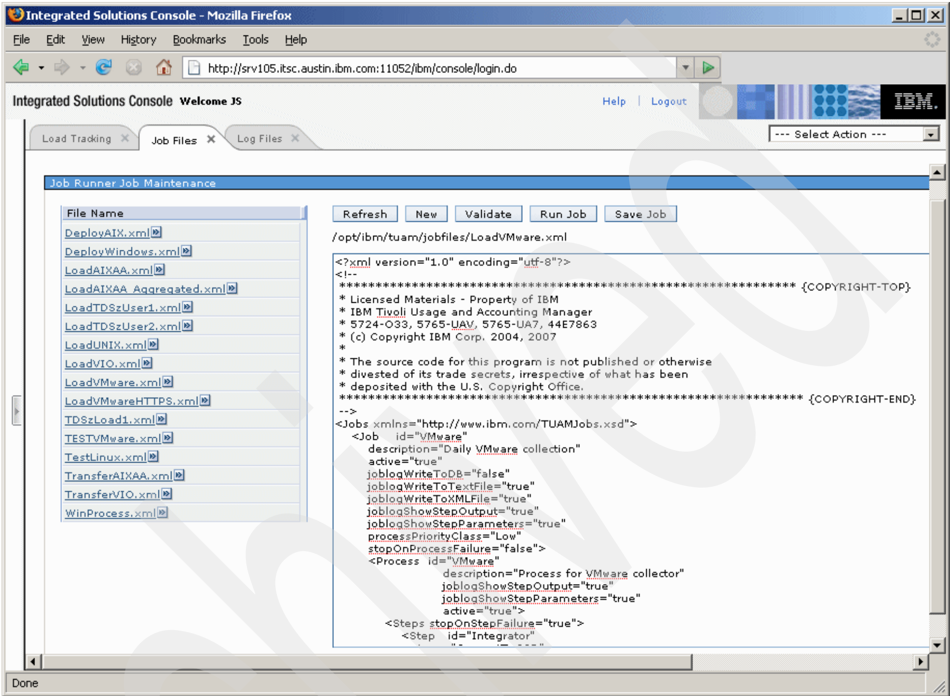


Figure 6-24 Run the LoadVMware.xml job from ISC

By default the JobRunner will use the previous day's date for executing the job. We can select any date (see Figure 6-25) to gather data for, as long as the VirtualCenter server keeps the aggregation level (the VMware default for 3600sec = 60 minutes is one month).

Note: If the VirtualCenter was not set up at least one day previously, remember to overwrite the default value preday with the current date for testing.

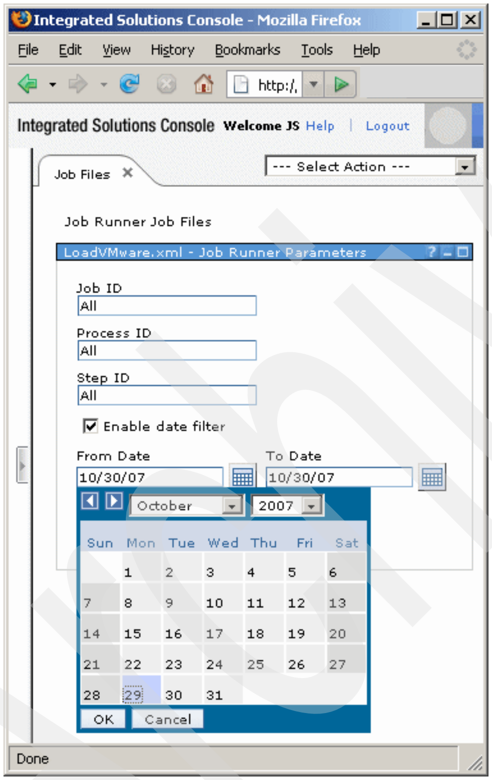


Figure 6-25 Select date and time for Job Runner on ISC

To check for the job logs, using the ISC, select **Usage and Accounting Manager → Chargeback Maintenance → Job Runner → Log Files** as shown in Figure 6-26.

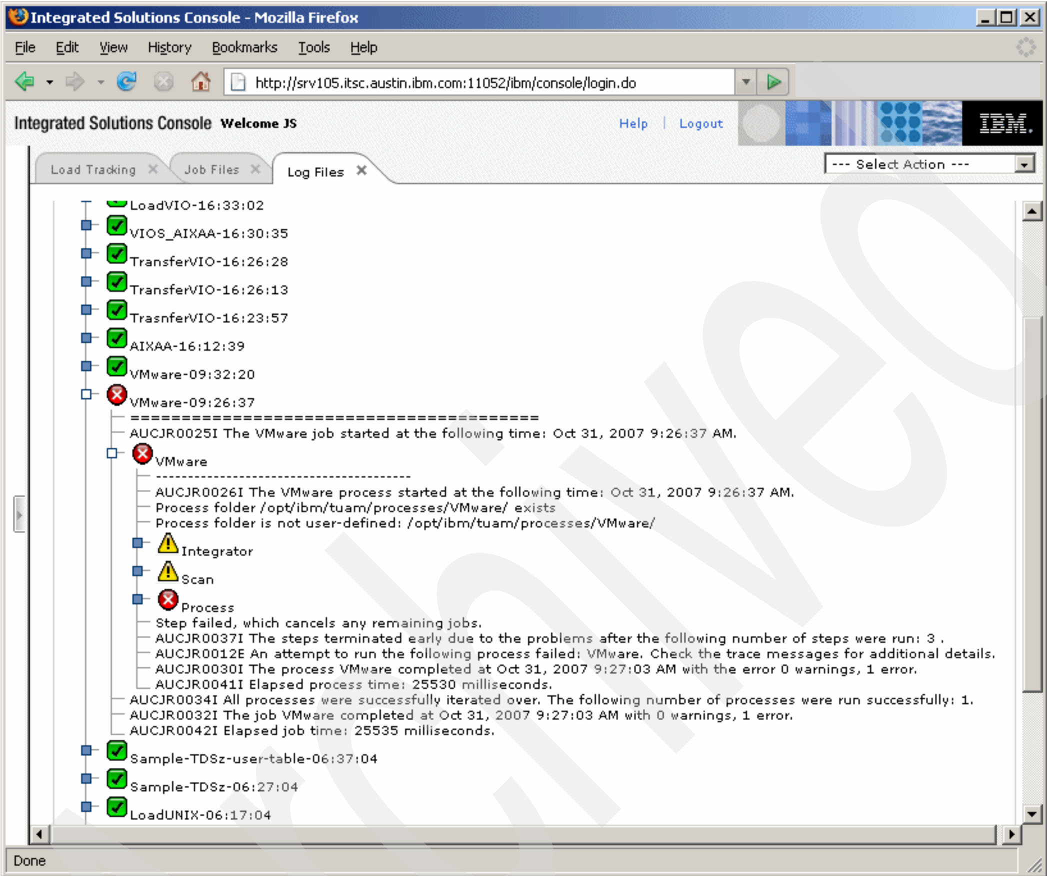


Figure 6-26 Checking the JobRunner log files via ISC

By clicking the symbols you can expand the detailed messages to locate the warning and error messages.

6.2.5 Reporting on VMware Accounting

By accessing the reporting server http://srv177/tuam, you can view reports on the VMware usage.

The summary for the previous week is shown in Figure 6-27 on page 199; the Top 10 for VMware charge backs are listed in Figure 6-28.

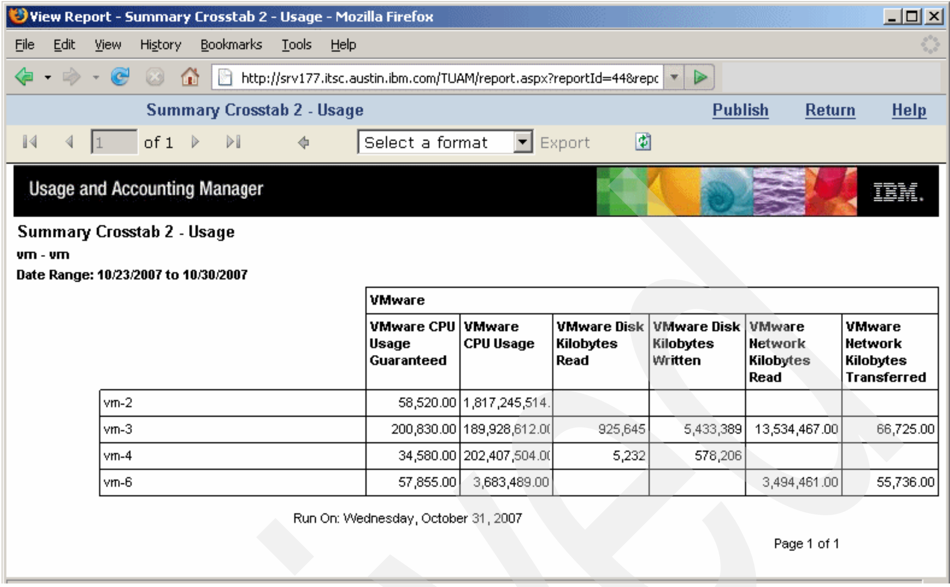


Figure 6-27 VMware summary for one week

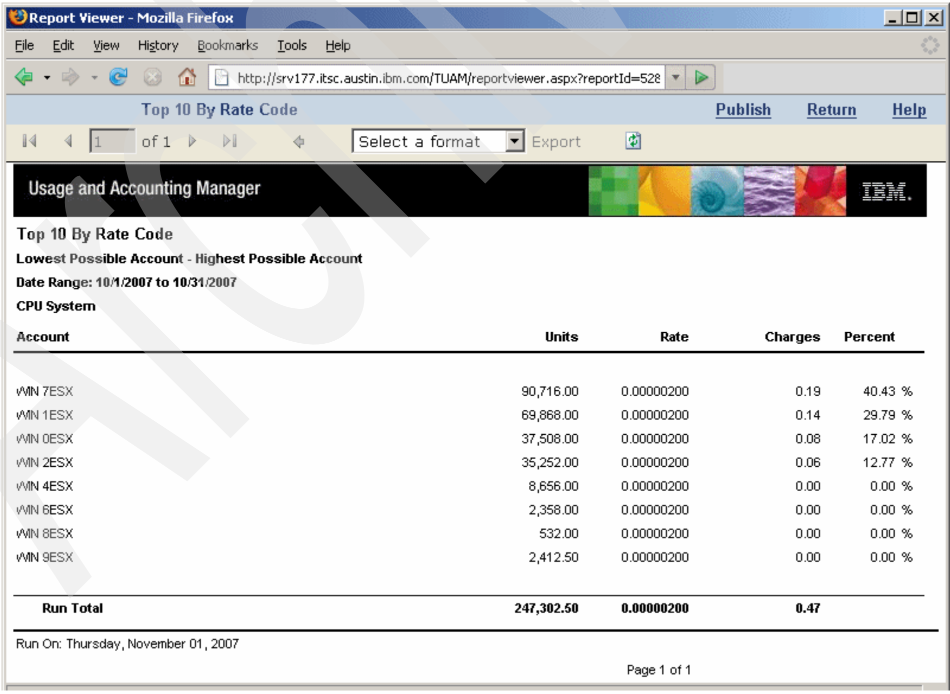


Figure 6-28 VMware Top 10 Rate Code Report

7

Processing data from Tivoli Decision Support for z/OS

In this chapter we discuss how Tivoli Decision Support for z/OS can be used as a data collector for Tivoli Usage and Accounting Manager. We assume that you already have a working Tivoli Decision Support for z/OS system, which you use for collecting data into standard or user-defined tables. This chapter covers the following topics:

7.1 Resource Accounting for z/OS Component

The Tivoli Usage and Accounting Manager uses the tables in the Resource Accounting for z/OS Component, which is part of Tivoli Decision Support for z/OS base. This section describes how to implement this feature and includes the following topics:

- 7.1.1, “Installing resource accounting” on page 202
- 7.1.2, “Populating lookup tables” on page 208
- 7.1.3, “Collecting z/OS accounting data” on page 214

7.1.1 Installing resource accounting

Our first step is to install the Resource Accounting for z/OS Component, commonly known as RAF component. Start the Tivoli Decision Support for z/OS dialog; it will appear as in Figure 7-1.

```
Options  Help
 ──────────────────────────────────────────────────────────────────────────
                         TDS for zOS Primary Menu
Command ===>

Select one of the following. Then press Enter.

              1.  Reports
              2.  Administration


+─────────────────────────────────────────────────────────────────────────────+
| Licensed Materials - Property of IBM (*) 5698-B06 (C) Copyright IBM Corp.   |
| 1993, 2007. All rights reserved. US Government Users Restricted Rights -    |
| Use, duplication or disclosure restricted by GSA ADP Schedule Contract with |
| IBM Corp. Terms that are trademarks are denoted by an asterisk (*) in Tivoli |
| Decision Support help panels. Trademarks are listed in the Product          |
| information option in the Help pull-down menu.                              |
+─────────────────────────────────────────────────────────────────────────────+
 F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-1 Tivoli Decision Support for z/OS Primary Menu

Select **Administration** → **Components** and find the Resource Accounting for z/OS Component as in Figure 7-2 on page 203.

```
Component  Space  Other  Help
──────────────────────────────────────────────────────────────────────────
                                Components                  Row 57 to 70 of 74
Command ===>

Select one or more components. Then press Enter to Open component.

/   Components                                       Status     Date
    OS/400 Job Statistics Component
    OS/400 Messages Component
    OS/400 Performance Component
    Resource Accounting for z/OS Component
    RACF Component
    Tivoli Information Management for z/OS(INFOMAN)
    Tivoli Storage Manager for z/OS (ADSM)
    Tivoli Workload Scheduler for z/OS (OPC)
    TCP/IP for z/OS Component
 F1=Help      F2=Split     F3=Exit      F5=New       F6=Install   F7=Bkwd
 F8=Fwd       F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-2 Tivoli Decision Support for z/OS component list

Your list of component may look different because it depends on which features of Tivoli Decision Support for z/OS you have and which components you have already installed. Select Resource Accounting for z/OS Component and press F6 to install it.

```
Component  Space  Other  Help
+---------------------------------------------------------------------------+
|                          RAF Component Parts             Row 1 to 12 of 12 |
| Command ===> ______________________________________________________________ |
|                                                                             |
| Select the parts of the component you want to install.  Then press Enter.  |
|                                                                             |
| /   Component Part                                       Status     Date    |
| _   CICS transaction accounting                                             |
| _   DASD space accounting                                                   |
| _   DB2 accounting                                                          |
| _   DFHSM backup accounting                                                 |
| _   DFHSM migration accounting                                              |
| _   IMS transaction accounting                                              |
| _   IMS Shared Queue transaction accounting                                 |
| s   Job accounting                                                          |
| _   NPM Session monitor accounting                                          |
| _   Session monitor accounting                                              |
| s   Started task accounting                                                 |
| s   TSO accounting                                                          |
| ***************************** Bottom of data ****************************** |
|  F1=Help     F2=Split    F7=Bkwd     F8=Fwd      F9=Swap    F12=Cancel     |
+---------------------------------------------------------------------------+
 F8=Fwd       F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-3 List of the resource accounting component parts

The resource accounting component is divided into parts, which allows you to install only the tables that you need for your reporting. In this scenario we will install the following parts:

- Job accounting
- Started task accounting
- TSO accounting

The tables belonging to those component parts are based mainly on data coming from SMF type 30 records. Select the component parts and press Enter.

```
 Component  Space  Other  Help
 s +-------------------------------------------------------------+ ----------
   |                    Installation Options                     |  to 70 of 74
 C |                                                             |
   | Select one of the following.  Then press Enter.             |
 S |                                                             |
   |     __ 1.  Online                                           |
 / |        2.  Batch                                            | te
   |                                                             |
   |  F1=Help     F2=Split    F6=Objects  F9=Swap    F12=Cancel  |
   +-------------------------------------------------------------+
 s   Resource Accounting for z/OS Component
     RACF Component
     Tivoli Information Management for z/OS(INFOMAN)
     Tivoli Storage Manager for z/OS (ADSM)
     Tivoli Workload Scheduler for z/OS (OPC)
     TCP/IP for z/OS Component
     UNIX Accounting Component
     UNIX Configuration Component
     UNIX Error Component
     UNIX Performance Component
     VM Accounting Component
  F1=Help      F2=Split     F3=Exit      F5=New       F6=Install   F7=Bkwd
  F8=Fwd       F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-4 Tivoli Decision Support for z/OS installation options

Here you have the option of doing the installation online or in batch. The online option lets your TSO session wait while the Tivoli Decision Support for z/OS objects are created; at the end, the list of lookup tables belonging to the component is displayed. The batch option will generate the JCL to create the Tivoli Decision Support for z/OS objects in batch. We used the online option to install the resource accounting component parts.

```
 Menu  Utilities  Compilers  Help
 ───────────────────────────────────────────────────────────────────────────────
 BROWSE    TIVO02.DRLOUT                                Line 00000000 Col 001 080
 Command ===>                                                    Scroll ===> PAGE
********************************* Top of Data **********************************
 DB2 Messages
SQL statements executed successfully
--------------------------------------------------------------------------------


Line       Log Collector Messages
--------------------------------------------------------------------------------
    1353
DRL0403I The purge condition for DRL.RAFJOBLOG is added.
    1358
DRL0403I The purge condition for DRL.RAFBATCH is added.
    1540
DRL0403I The purge condition for DRL.BILLED_DATA is added.
    1970
DRL0403I The purge condition for DRL.RAFADDRLOG is added.
    1975
DRL0403I Th +----------------------------------------------------+
    2231    | The parts you selected are installed successfully. |
  F1=Help   +----------------------------------------------------+  F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure 7-5 Resource accounting parts install messages

The messages from the resource accounting parts installation are presented in a browsable dataset. This dataset is created in the Tivoli Decision Support for z/OS dialog if it does not exist and will be reused in the next component installation. If you want to save the messages you can do so by renaming the *userid*.DRLOUT dataset (for example, to *userid*.DRLOUT.INSTRAF) after you have exited from the display shown in Figure 7-5. To continue, exit by pressing F3.

Figure 7-6 on page 207 shows the list of lookup tables that you might need to modify. A lookup table provides a value substitution in the data collection process.

```
Component  Space  Other  Help
+------------------------------------------------------------------------------+
|                              Lookup Tables              Row 1 to 10 of 10 |
| Command ===> ______________________________________________________________ |
|                                                                              |
| Select a lookup table.  Then press Enter to Edit the table in ISPF Edit     |
| mode.                                                                        |
|                                                                              |
| /   Lookup table                                                             |
| _   ACCOUNT                                                                  |
| _   ACCT_PRORATE                                                             |
| _   BILLING_PERIOD                                                           |
| _   CPU_NORMAL_DATA                                                          |
| _   CREDIT_DEBIT                                                             |
| _   CUSTOMER                                                                 |
| _   PRICE_LIST                                                               |
| _   RAFABATCH                                                              |
| _   RAFASTC                                                                  |
| _   RAFATSO                                                                  |
| ***************************** Bottom of data ****************************** |
|                                                                              |
|  F1=Help     F2=Split    F7=Bkwd     F8=Fwd      F9=Swap     F12=Cancel     |
+------------------------------------------------------------------------------+
 F8=Fwd       F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-6   Resource accounting lookup tables

Populating the lookup tables can be done online in the Tivoli Decision Support for z/OS dialog or in batch executing SQL INSERT statements. We used SQL commands; the details are presented in the next section. Press F3 to exit from the display of lookup tables. Figure 7-7 on page 208 indicates that the installation is complete.

```
Component  Space  Other  Help
──────────────────────────────────────────────────────────────────────────
                                Components                  Row 1 to 14 of 74
Command ===>

Select one or more components. Then press Enter to Open component.

/   Components                                       Status     Date
    Resource Accounting for z/OS Component           Installed  2007-10-10
    z/OS Performance Management (MVSPM) Component    Installed  2007-10-09
    z/OS System (MVS) Component                      Installed  2007-10-09
    DB2 Component                                    Installed  2007-10-09
    Sample Component                                 Installed  2007-10-09
    z/OS Availability Component
    z/OS Interval Job/Step Accounting Component
    z/VM Performance Component
    CICS Monitoring Component
    CICS Monitoring Partitioned Component
    CICS OMEGAMON Monitoring Component
    CICS Statistics Component
    +------------------------------------------------------------------+
    | Component 'Resource Accounting for z/OS Component' is installed. |
 F1 +------------------------------------------------------------------+
 F8=Fwd       F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-7 resource accounting component installed

7.1.2 Populating lookup tables

The reason we chose to use a batch job to populate the lookup tables is that this method affords easier backup and maintenance. When we have a batch job for this task it works like a backup copy of the table, and the table can easily be modified by changing the insert statements and rerunning the jobs.

When accounting and charge back is fully implemented in Tivoli Decision Support for z/OS, the summarized billing information is stored in the BILLED_DATA table. When we use the full accounting and charge back capabilities of the resource accounting component, all of the lookup tables must be populated. Here, we will only populate some of the tables because the table to feed Tivoli Usage and Accounting Manager is not BILLED_DATA. For the three parts of the resource accounting component we installed, Tivoli Usage and Accounting Manager uses data from the following tables:

- RAFADDRLOG

- RAFSTC
- RAFJOBLOG
- RAFBATCH
- RAFSESLOG
- RAFTSO

The information added in those tables, by use of lookup tables, is account code information and normalized CPU time. Thus the lookup tables we need to populate are:

- CPU_NORMAL_DATA
- RAFASTC
- RAFABATCH
- RAFATSO

The CPU_NORMAL_DATA lookup table

The CPU_NORMAL_DATA is used to calculate normalized CPU time so it will be possible to compare CPU consumption for tasks running in different systems. This table has columns for the system ID, dates when this entry is valid, the relative power, and a comment. Example 7-1 is a batch job to populate the CPU_NORMAL_DATA table.

Example 7-1 Batch job to populate CPU_NORMAL_DATA

```
//TIVO02IC JOB (TDSZ,180),LENNART,CLASS=A,MSGCLASS=X,
//         NOTIFY=&SYSUID,REGION=0M
//RUNLOG  EXEC PGM=DRLPLC,
// PARM=('SYSTEM=DB8D SYSPREFIX=DRLSYS &PREFIX=DRL',
// '&DATABASE=DRLDB &STOGROUP=DRLSG')
//STEPLIB  DD DISP=SHR,DSN=DRL180.SDRLLOAD
//         DD DISP=SHR,DSN=DB8D8.SDSNLOAD
//DRLOUT   DD SYSOUT=*
//DRLDUMP  DD SYSOUT=*
//DRLIN    DD *
SQL DELETE FROM &PREFIX.CPU_NORMAL_DATA ;
SQL INSERT INTO &PREFIX.CPU_NORMAL_DATA    VALUES
-- CP_SMF_ID, CP_POWER, CP_START_DATE, CP_END_DATE,  CP_DESC
  ('ZT01',    1.0,      '2000-01-01',  '2099-12-31', 'ZTEC SYSTEM 1') ;
SQL INSERT INTO &PREFIX.CPU_NORMAL_DATA    VALUES
  ('ZT02',    1.1,      '2000-01-01',  '2099-12-31', 'ZTEC SYSTEM 2') ;
SQL INSERT INTO &PREFIX.CPU_NORMAL_DATA    VALUES
  ('SC43',    0.8177,   '2000-01-01',  '2099-12-31', 'POC SYSTEM 43') ;
```

```
SQL INSERT INTO &PREFIX.CPU_NORMAL_DATA    VALUES
  ('SC47',    0.8177,   '2000-01-01',  '2099-12-31', 'POC SYSTEM 47') ;
SQL INSERT INTO &PREFIX.CPU_NORMAL_DATA    VALUES
  ('SC48',    0.8177,   '2000-01-01',  '2099-12-31', 'POC SYSTEM 48') ;
```

Note: The CP_POWER column is used to divide the CPU second column to have a "normalized" value for CPU seconds. Our interpretation of normalized CPU time is that it should be comparable between systems. This means the more powerful a processor is, the lower the CP_POWER value.

The column to hold the normalized CPU seconds in the tables RAFBATCH, RAFTSO, and RAFSTC is CPUNMSEC. This column is not used in the standard extraction of Tivoli Decision Support for z/OS data to load in Tivoli Usage and Accounting Manager. Doing the CPU normalization in Tivoli Decision Support for z/OS is not necessary from the Tivoli Usage and Accounting Manager point of view. It has been described here in case you want to make use of it in other situations. Tivoli Usage and Accounting Manager can do its own CPU normalization; this is discussed in 7.3.5, "CPU normalization example" on page 247.

The RAFASTC lookup table

The RAFASTC is used to assign an account code to the data in the RAFSTC table. The process in Tivoli Decision Support for z/OS to do this is called a LOOKUP function within an UPDATE definition. For the RAFSTC table, the lookup function derived the ACCTID field from existing identifiers such as system ID, jobname, userid, program name, and other fields. The matching can be an exact match or using wildcard characters. Example 7-2 is a batch job to populate the RAFASTC table. Here we look at only the first character in the jobname and assign account codes according to this.

Example 7-2 Batch job to populate RAFASTC

```
//TIVO02IS JOB (TDSZ,180),LENNART,CLASS=A,MSGCLASS=X,
//         NOTIFY=&SYSUID,REGION=0M
//RUNLOG  EXEC PGM=DRLPLC,
// PARM=('SYSTEM=DB8D SYSPREFIX=DRLSYS &PREFIX=DRL',
// '&DATABASE=DRLDB &STOGROUP=DRLSG')
//STEPLIB  DD DISP=SHR,DSN=DRL180.SDRLLOAD
//         DD DISP=SHR,DSN=DB8D8.SDSNLOAD
//DRLOUT   DD SYSOUT=*
//DRLDUMP  DD SYSOUT=*
//DRLIN    DD *
SQL DELETE FROM &PREFIX.RAFASTC       ;
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
```

```
--  ACCT1,   ACCT2,   ACCT3,   ACCT4,   ACCT5,   RACFGRP,   USERID,
--  JOBNAME, SYSID,   PGMNAME, USERFLD, ACCTID
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'A%',    '%',     '%',     '%',     'AAAAST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'B%',    '%',     '%',     '%',     'BBBBST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'C%',    '%',     '%',     '%',     'CCCCST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    '|%',    '%',     '%',     '%',     'DDDDST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    '|%',    '%',     '%',     '%',     'DDDDST02' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'I%',    '%',     '%',     '%',     'IIIIST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'T%',    '%',     '%',     '%',     'TTTTST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'W%',    '%',     '%',     '%',     'WWWWST01' );
SQL INSERT INTO &PREFIX.RAFASTC            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    '%',     '%',     '%',     '%',     'ZZZZST99' );
```

The RAFABATCH lookup table

The RAFABATCH table is used to assign an account code to the data in the RAFBATCH table and is similar to the RAFASTC table. Example 7-3 shows a batch job to populate the RAFABATCH table.

Example 7-3 Batch job to populate RAFABATCH

```
//TIVO02IB JOB (TDSZ,180),LENNART,CLASS=A,MSGCLASS=X,
//         NOTIFY=&SYSUID,REGION=0M
//RUNLOG  EXEC PGM=DRLPLC,
// PARM=('SYSTEM=DB8D SYSPREFIX=DRLSYS &PREFIX=DRL',
// '&DATABASE=DRLDB &STOGROUP=DRLSG')
//STEPLIB  DD DISP=SHR,DSN=DRL180.SDRLLOAD
//         DD DISP=SHR,DSN=DB8D8.SDSNLOAD
//DRLOUT   DD SYSOUT=*
```

```
//DRLDUMP  DD SYSOUT=*
//DRLIN    DD *
SQL DELETE FROM &PREFIX.RAFABATCH     ;
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
--  ACCT1,   ACCT2,   ACCT3,   ACCT4,   ACCT5,   RACFGRP,   USERID,
--  JOBNAME, SYSID,   PGMNAME, JOBCLASS,USERFLD, ACCTID
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'A%',    '%',     '%',     '%',     '%',     'AAAABA01' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'I%',    '%',     '%',     '%',     '%',     'AAAABA02' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'K%',    '%',     '%',     '%',     '%',     'KKKKBA01' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'L%',    '%',     '%',     '%',     '%',     'LLLLBA01' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'O%',    '%',     '%',     '%',     '%',     'OOOOBA01' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'P%',    '%',     '%',     '%',     '%',     'TTTTBA01' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'S%',    '%',     '%',     '%',     '%',     'TTTTBA02' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    'T%',    '%',     '%',     '%',     '%',     'TTTTBA03' );
SQL INSERT INTO &PREFIX.RAFABATCH          VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',       '%',
    '%',     '%',     '%',     '%',     '%',     'ZZZZBA99' );
```

The RAFATSO lookup table

The RAFATSO table is used to assign an account code to the data in the RAFTSO table and is similar to the RAFASTC table. Example 7-4 shows a batch job to populate the RAFATSO table.

Example 7-4 Batch job to populate RAFATSO

```
//TIV002IT JOB (TDSZ,180),LENNART,CLASS=A,MSGCLASS=X,
//         NOTIFY=&SYSUID,REGION=0M
//RUNLOG  EXEC PGM=DRLPLC,
// PARM=('SYSTEM=DB8D SYSPREFIX=DRLSYS &PREFIX=DRL',
```

```
// '&DATABASE=DRLDB &STOGROUP=DRLSG')
//STEPLIB  DD DISP=SHR,DSN=DRL180.SDRLLOAD
//         DD DISP=SHR,DSN=DB8D8.SDSNLOAD
//DRLOUT   DD SYSOUT=*
//DRLDUMP  DD SYSOUT=*
//DRLIN    DD *
SQL DELETE FROM &PREFIX.RAFATSO       ;
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
--  ACCT1,   ACCT2,   ACCT3,   ACCT4,   ACCT5,   RACFGRP,
--  USERID,  TERMID,  SYSID,   USERFLD, ACCTID
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'A%',    '%',     '%',     '%',     'AAAATS01' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
'I%',    '%',     '%',     '%',     'AAAATS02' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'K%',    '%',     '%',     '%',     'KKKKTS01' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'L%',    '%',     '%',     '%',     'LLLLTS01' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'O%',    '%',     '%',     '%',     'OOOOTS01' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'P%',    '%',     '%',     '%',     'TTTTTS01' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'S%',    '%',     '%',     '%',     'TTTTTS02' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    'T%',    '%',     '%',     '%',     'TTTTTS03' );
SQL INSERT INTO &PREFIX.RAFATSO            VALUES
  ( '%',     '%',     '%',     '%',     '%',     '%',
    '%',     '%',     '%',     '%',     'ZZZZTS99' );
```

Now that we have populated the necessary lookup tables for the resource accounting component parts we can continue with collection of data, which is the topic of the next section.

7.1.3 Collecting z/OS accounting data

We are now ready to collect data for the resource accounting component tables. When we installed the resource accounting component parts, the following summary tables were created:

- BILLED_DATA
- RAFADDRLOG
- RAFBATCH
- RAFJOBLOG
- RAFSESLOG
- RAFSTC
- RAFTSO
- USE_SUMMARY_D
- USE_SUMMARY_D2
- USE_SUMMARY_D3
- USE_SUMMARY_D4

The BILLED_DATA and the USE_SUMMARY% tables are not used by Tivoli Usage and Accounting Manager sample collection job. For this reason we will run a collect including only the RAF% tables. If the SMF data has already been collected for other tables, it is very likely you need to specify the REPROCESS parameter in the collect job. Example 7-5 is a job to collect SMF data into the resource accounting tables.

Example 7-5 Batch job to collect SMF data into resource accounting tables

```
//TIVO02CR JOB (TDSZ,180),LENNART,CLASS=A,MSGCLASS=X,
//         NOTIFY=&SYSUID,REGION=0M
//RUNLOG  EXEC PGM=DRLPLC,
// PARM=('SYSTEM=DB8D SYSPREFIX=DRLSYS &PREFIX=DRL',
// '&DATABASE=DRLDB &STOGROUP=DRLSG')
//STEPLIB  DD DISP=SHR,DSN=DRL180.SDRLLOAD
//         DD DISP=SHR,DSN=DB8D8.SDSNLOAD
//DRLIN    DD *
COLLECT SMF
  INCLUDE LIKE 'DRL.RAF%'
  REPROCESS
  COMMIT AFTER BUFFER FULL
  BUFFER SIZE 200 M;
//DRLOUT   DD SYSOUT=*
//DRLLOG   DD DISP=SHR,DSN=TIVO02.TESTSMF.ZTEC1
//DRLDUMP  DD SYSOUT=*
```

With the REPROCESS parameter specified in the collect job we get a return code of 4. The Tivoli Decision Support for z/OS log collector writes useful messages on DRLOUT, shown in Example 7-6.

Example 7-6 Tivoli Decision Support for z/OS log collector messages

```
COLLECT SMF
  INCLUDE LIKE 'DRL.RAF%'
  REPROCESS
  COMMIT AFTER BUFFER FULL
  BUFFER SIZE 200 |;
DRL0300I Collect started at 2007-10-10-18.32.57.
DRL0302I Processing TIVO02.TESTSMF.ZTEC1 on TST010  .
DRL0341I The first-record timestamp is 2007-10-08-01.00.01.060000.
DRL0304W The log data set is being reprocessed.
         Dataset Name: TIVO02.TESTSMF.ZTEC1
DRL0342I The last-record timestamp is 2007-10-08-13.27.08.220000.
DRL0310I A database update started after 372899 records due to end of log, at
         2007-10-10-18.33.03.
DRL0003I
DRL0315I Records read from the log or built by log procedure:
DRL0317I Record name        !    Number
DRL0318I -------------------!----------
DRL0319I SMF_006            !         0
DRL0319I SMF_025            !         0
DRL0319I SMF_026            !       118
DRL0319I SMF_030            !      8736
DRL0320I Unrecognized       !    364045
DRL0318I -------------------!----------
DRL0321I Total              !    372899
DRL0003I
DRL0323I                              --------Buffer------ ------Database-----
DRL0324I Table name                  !   Inserts   Updates   Inserts   Updates
DRL0325I ----------------------------!---------------------------------------
DRL0326I DRL     .RAFADDRLOG         !        63        16        63         0
DRL0326I DRL     .RAFBATCH           !        41        28        41         0
DRL0326I DRL     .RAFJOBLOG          !       114      1008       114         0
DRL0326I DRL     .RAFSESLOG          !        20        49        20         0
DRL0326I DRL     .RAFSTC             !        25        38        25         0
DRL0326I DRL     .RAFTSO             !         9        11         9         0
DRL0325I ----------------------------!---------------------------------------
DRL0327I Total                       !       272      1150       272         0
DRL0003I
DRL0301I Collect ended at 2007-10-10-18.33.04.
DRL0356I To update the database, the algorithm insert was most selected
```

We can see from the log collector messages that four types of SMF records are used to update the resource accounting tables. In our SMF dataset type 6 and 25 are missing, meaning that there will be no information about printing in our resource accounting tables. There are also messages for the tables included in this collect, and how many rows were inserted and updated. The number of updates is zero for all tables because the tables were empty from the start and the buffer of 200 MB was large enough to hold all of the processed SMF records.

> **Note:** Do not use the REPROCESS parameter in your Tivoli Decision Support for z/OS production jobs. In a production environment the Tivoli Decision Support for z/OS collect jobs should also include the resource accounting tables by adding LIKE ‘DRL.RAF%’ to the INCLUDE parameter.

7.2 The z/OS resource accounting data

This section discusses our customization in Tivoli Usage and Accounting Manager for accessing z/OS resource accounting data. The discussion consists of:

- 7.2.1, “Establishing connectivity to DB2 for z/OS” on page 216
- 7.2.2, “Loading the resource accounting data” on page 220
- 7.2.3, “Correcting a Rate Code not defined situation” on page 227

7.2.1 Establishing connectivity to DB2 for z/OS

Our processing server must be able to access Tivoli Decision Support for z/OS database in DB2 for z/OS.

We need to know certain things about DB2 on z/OS in order to be able to connect our DB2 in Linux systems to the DB2 on z/OS. During DB2 startup in z/OS, the DSNL004I message is written to a system log that has information about LOCATION, DOMAIN, and TCPPORT of the DB2 subsystem. Example 7-7 shows our DSNL004I message.

Example 7-7 DDF start messages in z/OS syslog

```
00000090  DSNL519I  -DB8D DSNLILNR TCP/IP SERVICES AVAILABLE 334
00000090             FOR DOMAIN wtsc67.itso.ibm.com AND PORT 38030
00000090  DSNL004I  -DB8D DDF START COMPLETE 335
00000090             LOCATION  DB8D
00000090             LU        USIBMSC.SCPDB8D
00000090             GENERICLU -NONE
00000090             DOMAIN    wtsc67.itso.ibm.com
```

```
00000090            TCPPORT   38030
00000090            RESPORT   38031
00000090  DSN9022I  -DB8D DSNYASCP 'START DB2' NORMAL COMPLETION
```

We need access to the DB2 for z/OS jar file, the db2jcc_license_cisuz.jar. This file is available with the DB2 Connect™ product. This jar file is also available from the DB2 for z/OS product in the HFS. Our DB2 for z/OS HFS is located in the /usr/lpp/db2 directory structure. Example 7-8 shows the sequence of commands in Linux to download the files. We need the db2jcc_license_cisuz.jar and db2jcc.jar.

Example 7-8 Copy the DB2 z/OS JDBC drivers to Linux

```
[root@srv105 ~]# mkdir /opt/dbzos
[root@srv105 ~]# cd /opt/dbzos
[root@srv105 dbzos]# ftp wtsc67.itso.ibm.com
Connected to wtsc67.itso.ibm.com.
220-FTPMVS1 IBM FTP CS V1R7 at wtsc67.itso.ibm.com, 21:21:11 on 2007-10-11.
220 Connection will close if idle for more than 5 minutes.
534 Server is not willing to accept security mechanism
504 Server does not understand the specified mechanism
KERBEROS_V4 rejected as an authentication type
Name (wtsc67.itso.ibm.com:root): tivo02
331 Send password please.
Password:
230 TIVO02 is logged on.  Working directory is "TIVO02.".
Remote system type is MVS.
ftp> cd /usr/lpp/db2/db8d/jcc/classes
250 HFS directory /usr/lpp/db2/db8d/jcc/classes is the current working directory
ftp> ls
227 Entering Passive Mode (9,12,4,22,5,2)
125 List started OK
total 5416
drwxr-xr-x   2 HAIMO    SYS1        8192 Oct 13  2003 IBM
-rwxr-xr-x   2 HAIMO    SYS1     1213282 Jan 15  2007 db2jcc.jar
-rwxr-xr-x   2 HAIMO    SYS1       23709 Jan 15  2007 db2jcc_javax.jar
-rwxr-xr-x   2 HAIMO    SYS1        2063 Jan 15  2007 db2jcc_license_cisuz.jar
-rwxr-xr-x   2 HAIMO    SYS1     1517274 Jan 15  2007 sqlj.zip
250 List completed successfully.
ftp> bin
200 Representation type is Image
ftp> mget *.jar
mget db2jcc.jar? y
227 Entering Passive Mode (9,12,4,22,5,4)
125 Sending data set /usr/lpp/db2/db8d/jcc/classes/db2jcc.jar
```

```
250 Transfer completed successfully.
1213282 bytes received in 3.1 seconds (3.8e+02 Kbytes/s)
mget db2jcc_javax.jar? n
mget db2jcc_license_cisuz.jar? y
227 Entering Passive Mode (9,12,4,22,5,5)
125 Sending data set /usr/lpp/db2/db8d/jcc/classes/db2jcc_license_cisuz.jar
250 Transfer completed successfully.
2063 bytes received in 0.0061 seconds (3.3e+02 Kbytes/s)
ftp> quit
221 Quit command received. Goodbye.
```

The next step is to define the DB2 driver for z/OS to Tivoli Usage and Accounting Manager. Start the Integrated Solutions Console, and select **Usage and Accounting Manager** → **System Maintenance** → **Configuration** → **Drivers**. Click the **New** button and add the driver as shown in Figure 7-8.

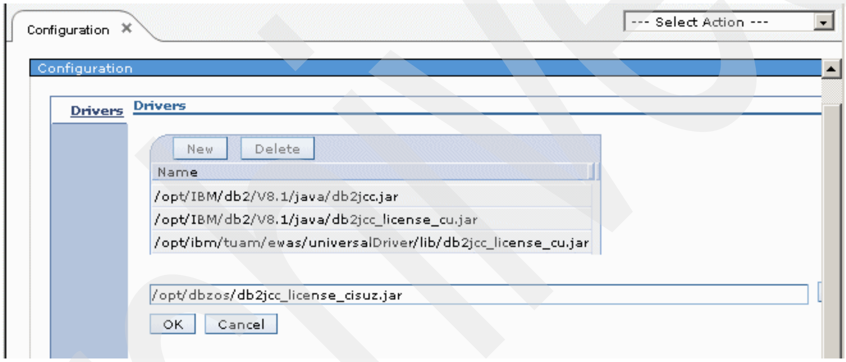


Figure 7-8 Adding the DB2 z/OS driver

After adding the new driver we need to restart the WebSphere Application Server.

We continue with adding DB2 z/OS as a data source in Tivoli Usage and Accounting Manager. In the ISC, select **Usage and Accounting Manager** → **System Maintenance** → **Configuration** → **Data Sources.** Select the Collector - Database tab, click the **New** button, and add the data source as shown in Figure 7-9 on page 219.

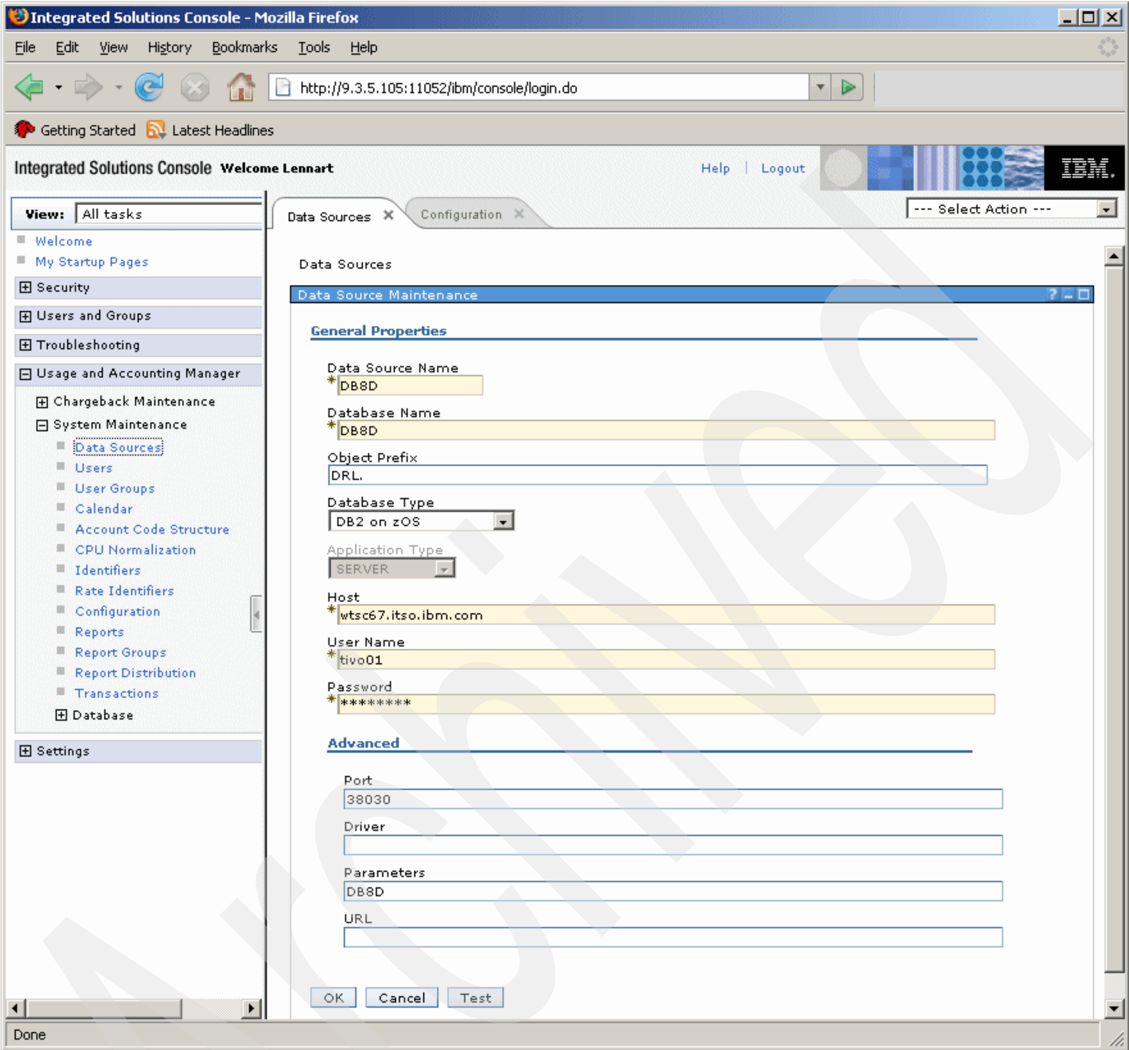


Figure 7-9 Add DB2 z/OS as a data source

In Figure 7-9, the parameters are taken from the DSNL004I message in Example 7-7 on page 216:

- Location name is specified as Database Name, and in the Advanced section, within the Parameters field.
- Port is provided in the Advanced section within the Port field.
- Domain is provided as the Host that the DB2 for z/OS resides on.

You can now check the connectivity using the **Test** button. Figure 7-10 shows that the connection was successful.

Figure 7-10 Test connection to DB2 z/OS

7.2.2 Loading the resource accounting data

In this section, we describe how the supplied sample job file can be used to load the data that was collected by Tivoli Decision Support for z/OS into Tivoli Usage and Accounting Manager. A sample job file called SampleTDSz.xml is provided. You can copy the sample file to build your own load job. We called our job TDSzLoad1.xml. Figure 7-11 on page 221 shows our TDSzLoad1.xml job.

A complete listing of our modified job is provided in "Sample job for Tivoli Decision Support for z/OS" on page 321.

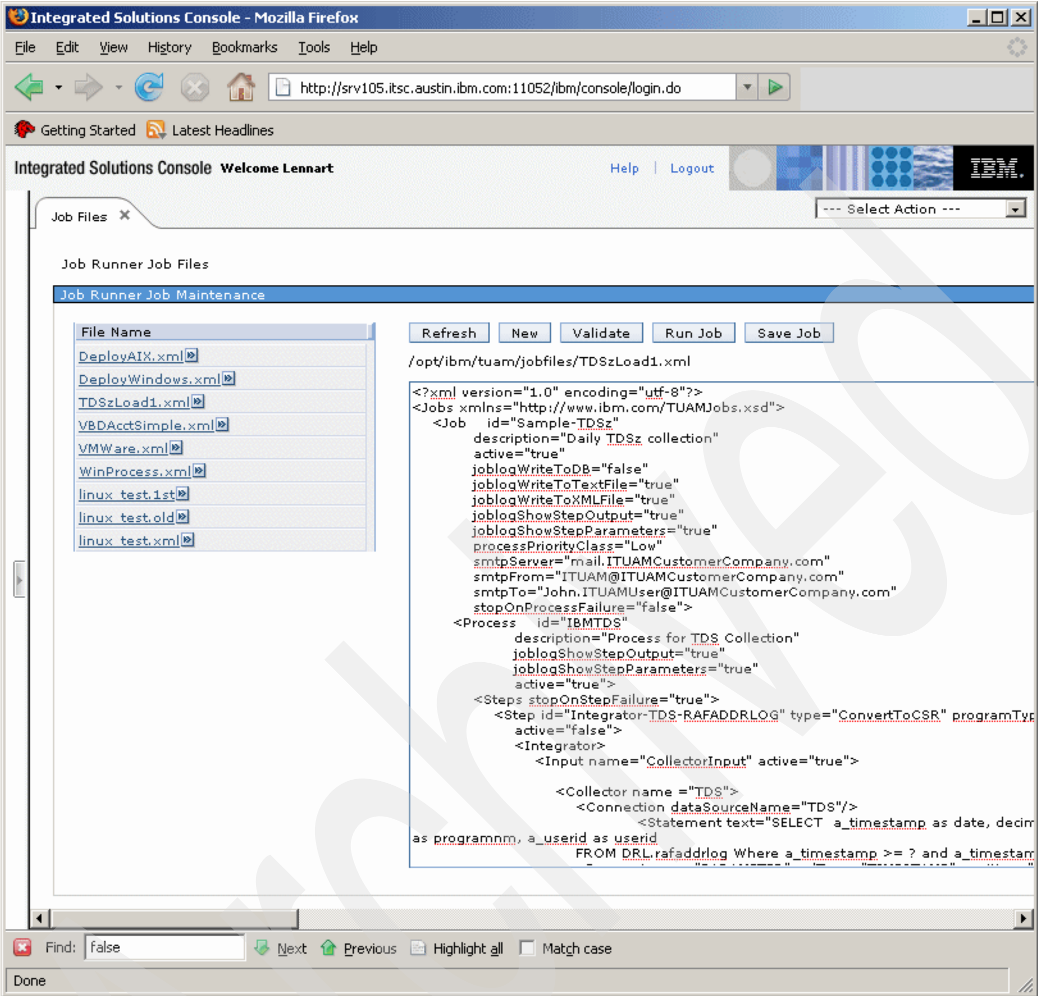


Figure 7-11 The TDSzLoad1.xml file

The job file has one process with the ID of IBMTDS. This process starts with 15 steps for executing the Tivoli Usage and Accounting Manager integrator, one for each Tivoli Decision Support for z/OS resource accounting table. We have only collected data for the following resource accounting tables:

- RAFADDRLOG
- RAFBATCH
- RAFJOBLOG
- RAFSESL
- RAFSTC
- RAFTSO

In each of the steps to process those six tables there are some things that have to be changed:

- Setting `active="true"` on the appropriate step directive
- Setting `dataSourceName="DB8D"` which is the data source we defined as described in Figure 7-9 on page 219.

In Figure 7-12 we have highlighted the changed information.

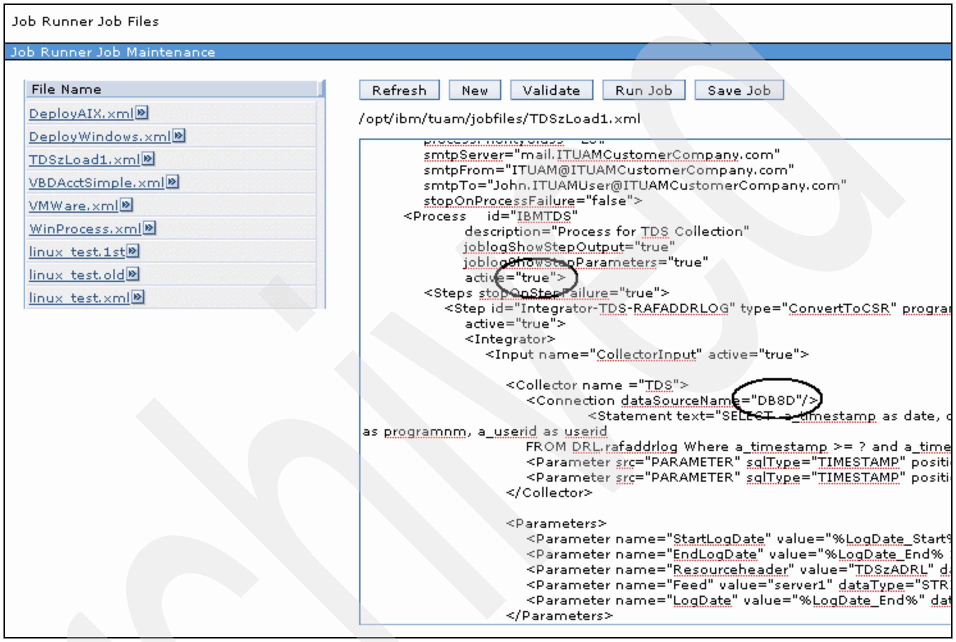


Figure 7-12 Changes in TDSzLoad1.xml file

> **Note:** You may find it more convenient to use another editor than the one built into ISC. The name of the job file appears below the buttons in Figure 7-12. Make sure you change active from `false` to `true` only for the steps you want to execute.

Once the changes are made, click the **Validate** button and then the **Save Job** button. You get a message like the one in Figure 7-13 when the job file is free of syntax errors.

Figure 7-13 TDSzLoad1.xml verified

A successful validation does not guarantee that the job will run without errors. There may be errors caused by pointing to the wrong data source, or other logical errors. To run the job click the **Run Job** button and select the dates to load into Tivoli Usage and Accounting Manager, as shown in Figure 7-14.

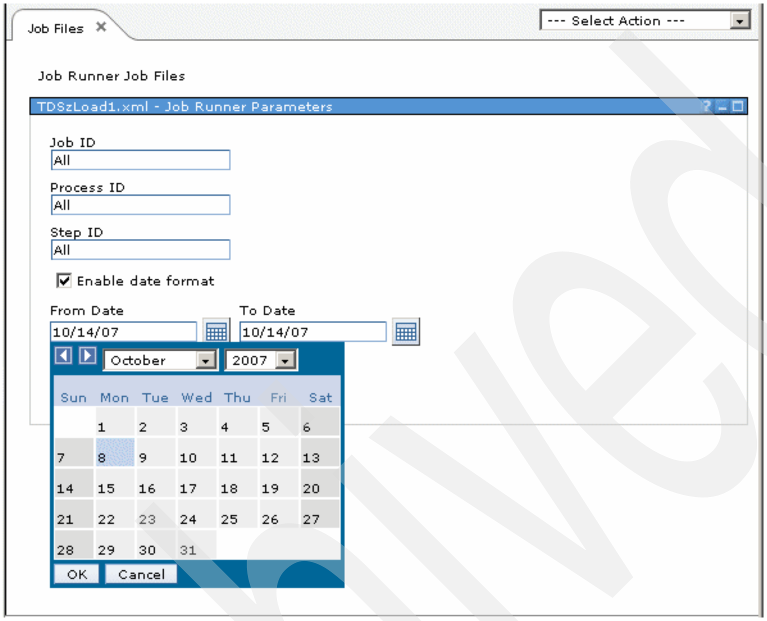


Figure 7-14 Specify dates using calender

You can use the calender button to select the dates. If a range of two ore more days is selected the job will be run multiple times. When selection is done click the **OK** button.

Typically, you will get the message indicating that the job ran OK, with a warning, or failed. Here, we have a warning, as shown in Figure 7-15 on page 224.

Figure 7-15   Job ran with warning

We should now investigate the cause of this warning message. Tivoli Usage and Accounting Manager shows the log of the execution of the job. The information is found in the ISC by selecting **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Log Files.** The results are shown in Figure 7-16.

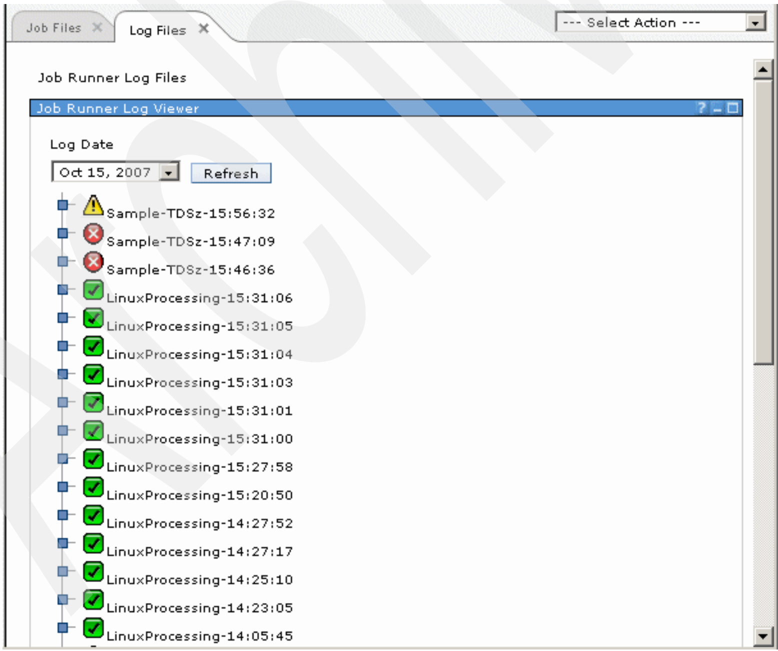


Figure 7-16   Job Runner Log Files

To see the details of what happened, we drill down in the log information and see that all of the six resource accounting tables where successfully processed, but there is a warning on the Process step. This is displayed in Figure 7-17.

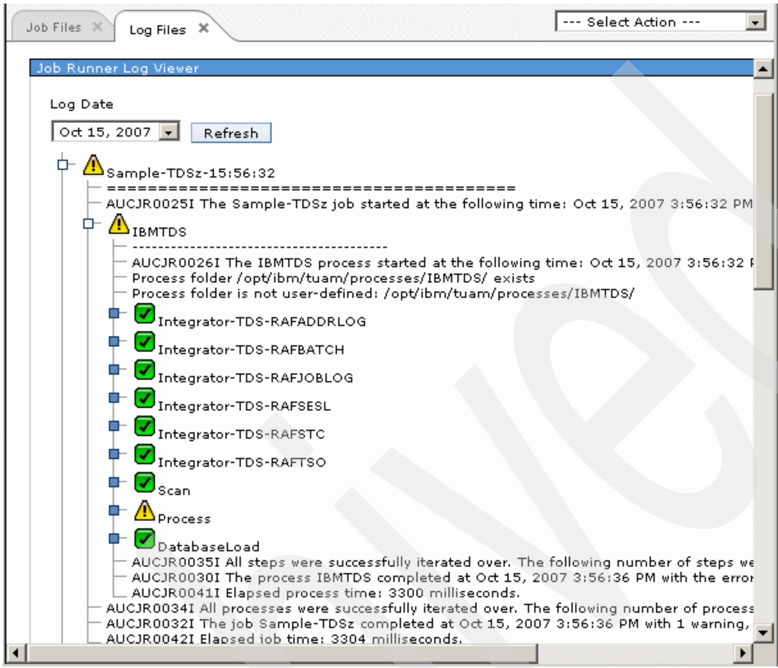


Figure 7-17 Log File drill-down

One step further in the drill down will show the cause of the warning message (Figure 7-18 on page 226).

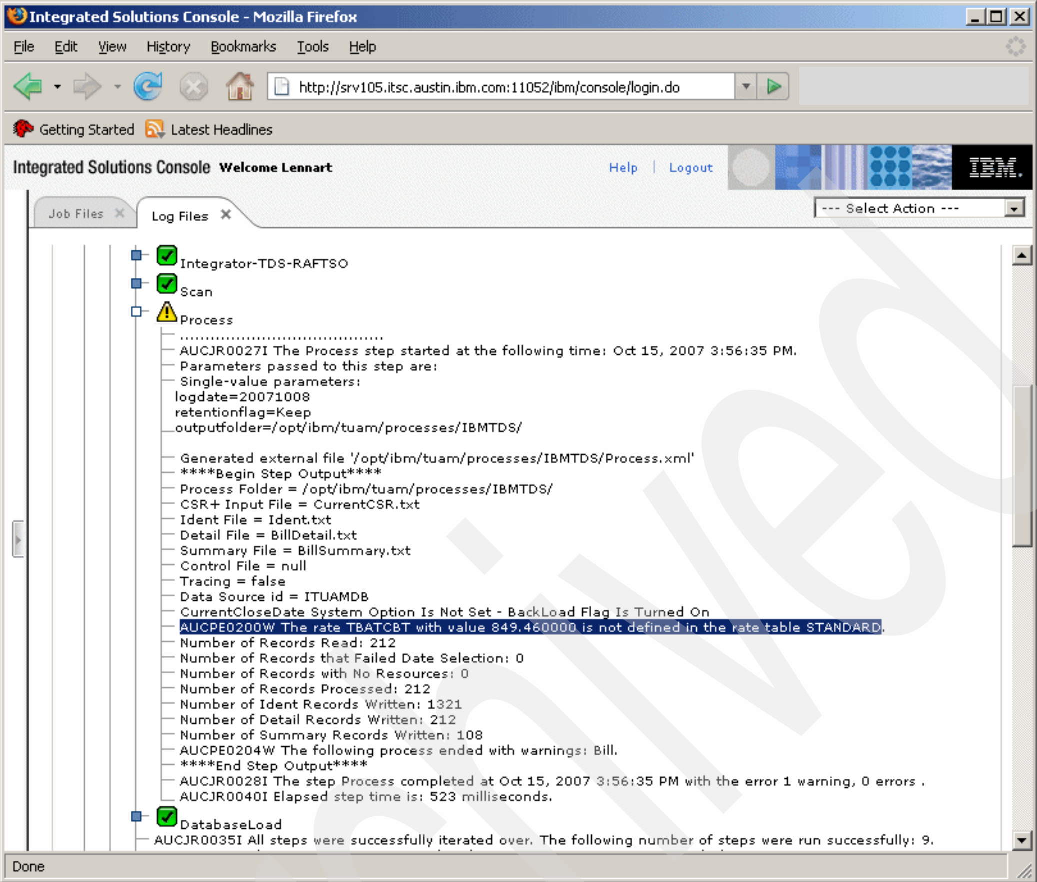


Figure 7-18 Log File warning message

It is easy to understand from the warning message what is wrong. There is a Rate Code missing in the STANDARD Rate Table. This is a problem that will not be uncommon when we implement Tivoli Usage and Accounting Manager in a production environment and start looking at what measures to charge. Now we go back to the TDSzLoad1.xml. Look at the step “Integrator-TDS-BATCH” in the OutputFields section shown in Example 7-9.

Example 7-9 TDSzLoad1.xml OutputFields for RAFBATCH

```
<OutputFields>
   <OutputField name="Feed" src="PARAMETER" srcName="Feed" />
   <OutputField name="headerstartdate" src="INPUT" srcName="14" />
   <OutputField name="headerenddate" src="INPUT" srcName="14" />
   <OutputField name="ACCOUNT" src="INPUT" srcName="13" />
   <OutputField name="PERIOD" src="INPUT" srcName="11" />
```

```
    <OutputField name="DATE" src="INPUT" srcName="14" />
    <OutputField name="JOBNAME" src="INPUT" srcName="10" />
    <OutputField name="PRINTER" src="INPUT" srcName="15" />
    <OutputField name="Z003" src="INPUT" srcName="1" resource="true" />
    <OutputField name="Z001" src="INPUT" srcName="2" resource="true" />
    <OutputField name="Z017" src="INPUT" srcName="3" resource="true" />
    <OutputField name="Z016" src="INPUT" srcName="4" resource="true" />
    <OutputField name="TBASRBT" src="INPUT" srcName="5" resource="true" />
    <OutputField name="TBATCBT" src="INPUT" srcName="6" resource="true" />
    <OutputField name="Z006" src="INPUT" srcName="8" resource="true" />
    <OutputField name="Z007" src="INPUT" srcName="9" resource="true" />
    <OutputField name="TBAEXCP" src="INPUT" srcName="7" resource="true" />
</OutputFields>
```

Here we see the use of the resource named TBATCBT, which is used for the TCBTIME column in the RAFBATCH table. This resource name is used for matching with the Rate in the rate table.

7.2.3 Correcting a Rate Code not defined situation

Because the resource TBATCBT is correct, we must add a new rate code and rerun the loading of Tivoli Decision Support for z/OS data.

Adding a new Rate Code

Start the ISC and select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Rates.** A screen like Figure 7-19 on page 228 will be displayed.

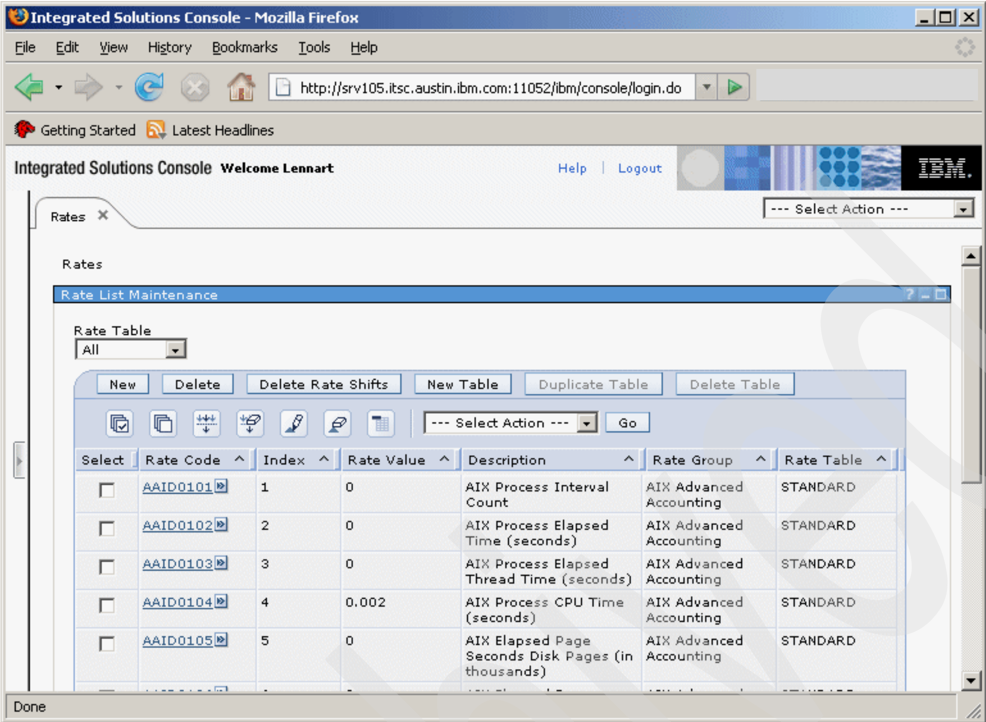


Figure 7-19 Rates

Click the **New** button to define a new Rate Code. The Rate Code is the only required field, but we will fill in some of the fields as shown in Figure 7-20 on page 229.

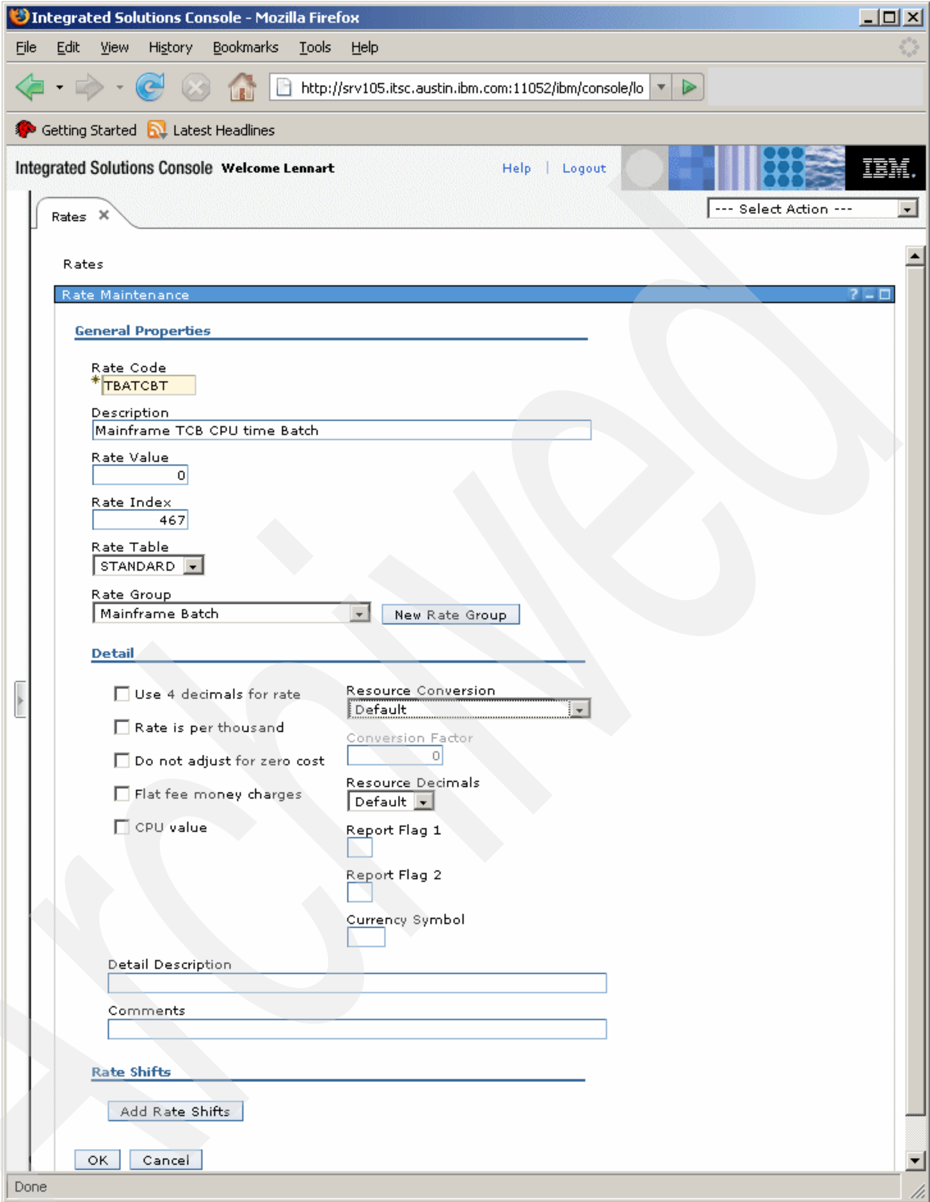


Figure 7-20 Rate Code TBATCBT

Click the **OK** button to add the new Rate Code. There is no message issued, so to verify that TBATCBT has been added, select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Rates Codes** and expand the Mainframe Batch group as in Figure 7-21 on page 230.

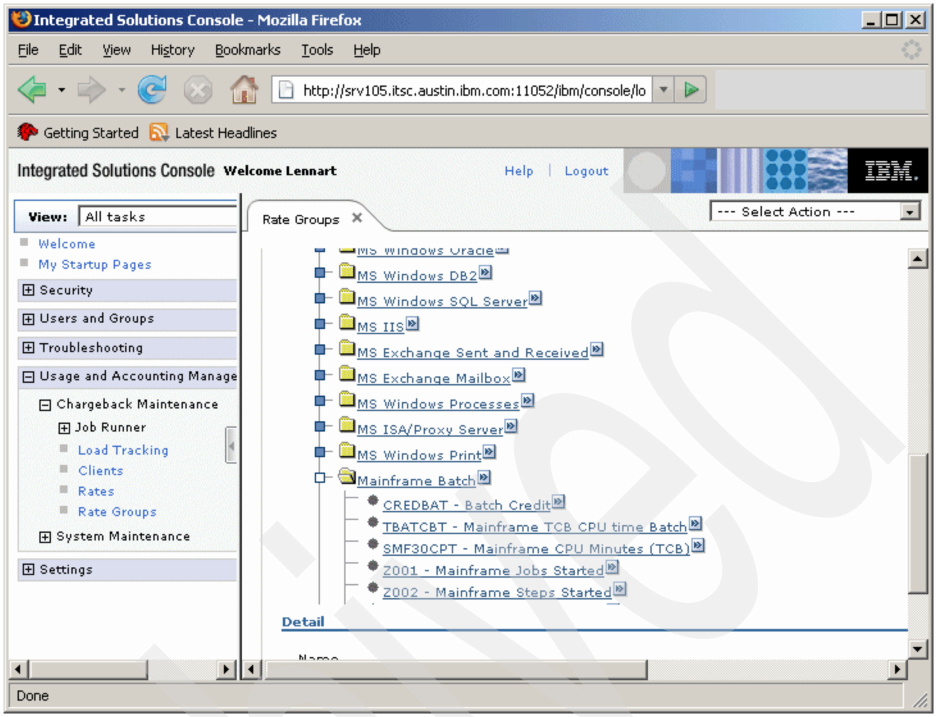


Figure 7-21 Mainframe Batch Rate Group

Note: There is already a Rate Code defined for CPU TCB time, SMF30CPT, which probably can be used instead of adding the new TBATCBT.

Re-run the Tivoli Decision Support for z/OS job file

If we try to run the TDSzLoad1.xml Job File again the Database Load step will fail. Tivoli Usage and Accounting Manager keeps track of the data loaded and will not allow the same data to be loaded twice. Therefore, to re-run the TDSzLoad1.xml Job File we first need to delete the old data. To do that, start the ISC and select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Load Tracking**. A panel like the one in Figure 7-22 on page 231 will display.

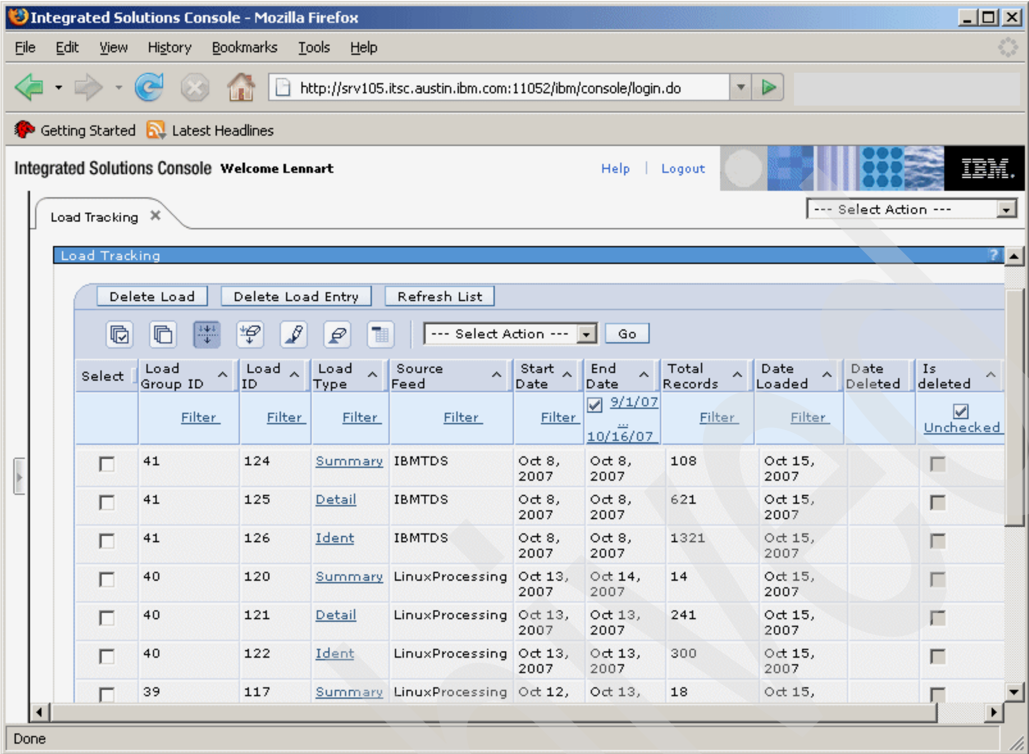


Figure 7-22 Load Tracking

We looked at the following columns:

- Source Feed
- Start Date
- End Date
- Total Records
- Date Loaded

We determined that the first three entries were the ones we needed to delete.

> **Note:** There are two delete buttons. **Delete Load** deletes the loaded data; the **Delete Load Entry** only deletes the entries in the Load Tracking, after all load data has been deleted.

Check the **Select** box for the identified entries and click the **Delete Load** button as shown in Figure 7-23 on page 232.

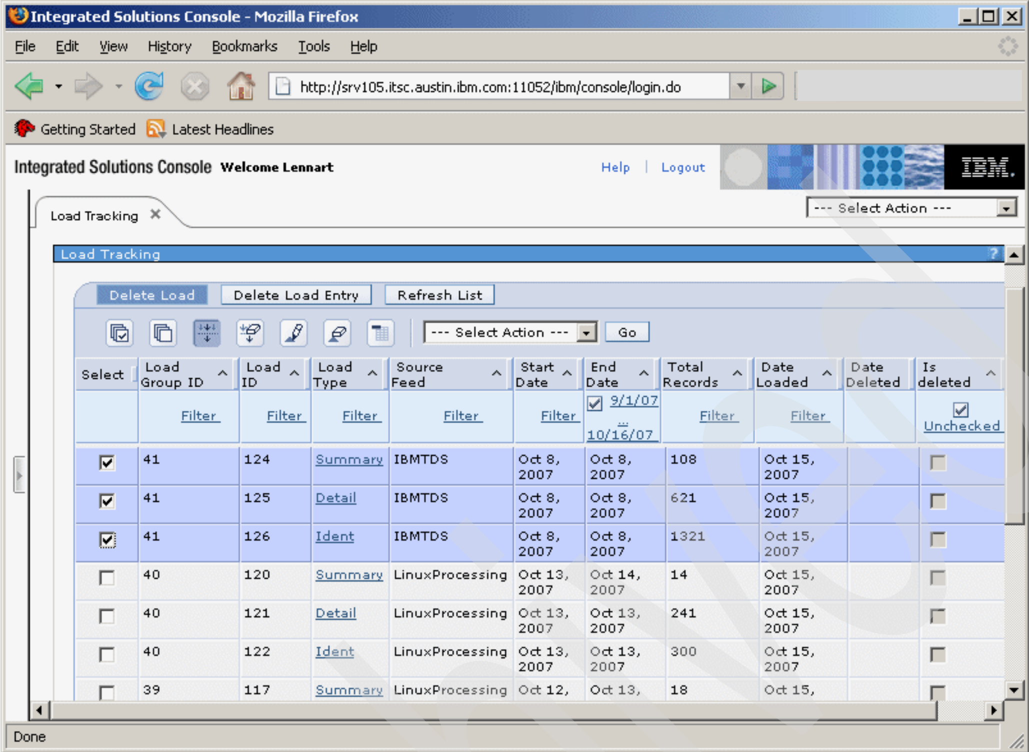


Figure 7-23   Delete Load

Confirm the Delete Load by clicking the **Yes** button shown in Figure 7-24.

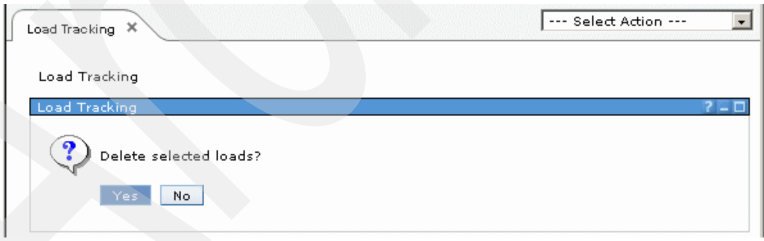


Figure 7-24   Confirm Delete Load

When the previously loaded data has been deleted we can re-run the TDSzLoad1.xml Job File. Select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files.** View the pop-up menu by clicking the icon after TDSzLoad1.xml (Figure 7-25 on page 233).

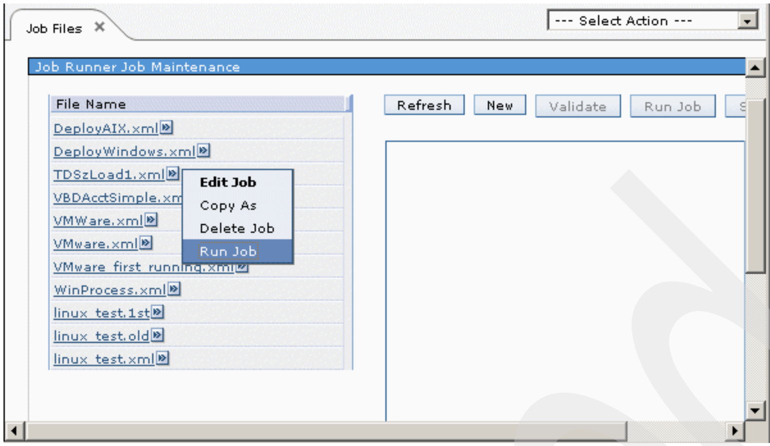


Figure 7-25 Popup menu for TDSzLoad1.xml

Click the Run Job option, specify the From Date and To Date, and run the job. Now you should receive a message that the job ran successfully (Figure 7-26).

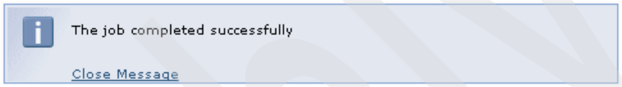


Figure 7-26 Job success notice

7.3 Defining z/OS user-defined data

In this section we describe how to load data from sources other than the resource accounting tables. This will be a common scenario at most installations where Tivoli Decision Support for z/OS has been customized, or when you find data in some of the other Tivoli Decision Support for z/OS tables useful for accounting. The topics covered here are:

- 7.3.1, “Creating a sample view” on page 233
- 7.3.2, “Loading user-defined data” on page 236
- 7.3.3, “Defining rate codes” on page 240
- 7.3.4, “Account code translation example” on page 243
- 7.3.5, “CPU normalization example” on page 247

7.3.1 Creating a sample view

We start by creating a view that will be used as user-defined data to be loaded into Tivoli Usage and Accounting Manager. Example 7-10 on page 234 shows statements for the Tivoli Decision Support for z/OS log collector to create this sample view. The MVS_ADDRSPACE_T table has information about z/OS address spaces and is a good source for accounting data. In our sample view we have picked some of the columns that could be of interest when accounting for z/OS usage.

Example 7-10 Sample view

```
--SQL DROP   VIEW &PREFIX.SAMPLE_ITUAM_V   ;
SQL CREATE VIEW &PREFIX.SAMPLE_ITUAM_V
  (DATE_START         ,
   DATE_END           ,
   MVS_SYSTEM_ID      ,
   SUBSYSTEM_ID       ,
   JOB_NAME           ,
   ACCOUNT_FIELD1     ,
   ELAPSED_HOURS      ,
   CPU_SECONDS        ,
   ZAAP_SECONDS       ,
   ZIIP_SECONDS       ,
   IO_SERVICE_UNITS   ,
   JOB_COUNT          )
  AS
 SELECT
   DATE(JOB_START_TIME) ,
   DATE(JOB_END_TIME)   ,
   MVS_SYSTEM_ID        ,
   SUBSYSTEM_ID         ,
   JOB_NAME             ,
   ACCOUNT_FIELD1       ,
   SUM(ELAPSED_SECONDS/3600) ,
   SUM(CPU_TOTAL_SECONDS)    ,
   SUM(IFA_CPU_SECONDS)      ,
   SUM(ZIIP_CPU_SECONDS)     ,
   SUM(SERVICE_UNITS_IO)     ,
   COUNT(*)
  FROM &PREFIX.MVS_ADDRSPACE_T
  WHERE JOB_START_TIME IS NOT NULL
    AND JOB_END_TIME IS NOT NULL
  GROUP BY
    DATE(JOB_START_TIME)  ,
    DATE(JOB_END_TIME)    ,
    MVS_SYSTEM_ID   ,
    SUBSYSTEM_ID    ,
    JOB_NAME        ,
```

```
  ACCOUNT_FIELD1
 ;
```

Use the Tivoli Decision Support for z/OS dialog to call the log collector for creating the view. Save the statements in a member in the Tivoli Decision Support for z/OS LOCAL.DEFS dataset. Start the Tivoli Decision Support for z/OS dialog and select **Administration** → **Other** → **Process TDS for zOS statements** and a screen like Figure 7-27 is returned.

```
 Other  Utilities  Help
+------------------------------------------------------------------------+
|                      Process TDS for zOS Statements                  |
|                                                                        |
| Type in the data set name. Then press Enter to edit the statements.  |
|                                                                        |
| Input data set name  DRL180.LOCAL.DEFS(ADDRSPV1)                      |
|                                                                        |
| Type of statements  . . . . 1   1.  Log collector                     |
|                                 2.  Report definition                 |
|                                                                        |
| Show input statements . . . 2   1.  Yes                               |
|                                 2.  No                                  |
|                                                                        |
| Trace SQL calls . . . . . . 2   1.  Yes                               |
|                                 2.  No                                  |
|                                                                        |
|  F1=Help     F2=Split    F5=Execute  F9=Swap    F12=Cancel            |
+------------------------------------------------------------------------+



  F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 7-27 Process Tivoli Decision Support for z/OS statements

For Input data set name type the name of the LOCAL.DEFS with the member name. Set Type of statements to 1 and press F5 to execute the statements. The Tivoli Decision Support for z/OS dialog browses the DRLOUT dataset where the SQL messages are displayed. We have created a view and can now look at its content. In the Tivoli Decision Support for z/OS dialog select **Administration** → **Tables** and find the SAMPLE_ITUAM_V view. Select the view and press F11 to display the contents, as in Figure 7-28 on page 236.

```
 Menu  Utilities  Compilers  Help
 ________________________________________________________________________________
 BROWSE    TIVO02.DRLTAB                                Line 00000000 Col 001 080
 Command ===>                                                     Scroll ===> PAGE
******************************** Top of Data *********************************
 DATE       DATE       MVS    SUBSYSTEM JOB      ACCOUNT  ELAPSED
 START      END        SYSTEM ID        NAME     FIELD1   HOURS
                       ID
 ---------- ---------- ------ --------- -------- -------- ----------------------
 2007-10-10 2007-10-10 ZT02   OMVS      PIERRE8  SYS0000  +.119444444444444E-03
 2007-10-10 2007-10-10 ZT02   OMVS      PIERRE9  SYS0000  +.646111111111111E-02
 2007-10-10 2007-10-10 ZT02   STC       BPXAS    -        +.756997500000000E+01
 2007-10-10 2007-10-10 ZT02   STC       IMA1FDBR -        +.222222222222222E-04
 2007-10-10 2007-10-10 ZT02   STC       IMA2RDR  -        +.222222222222222E-04
 2007-10-10 2007-10-10 ZT02   TSO       AJRW1    SYS0000  +.114319722222222E+01
 2007-10-10 2007-10-10 ZT02   TSO       PIERRE   SYS0000  +.381188333333333E+01
 2007-10-10 2007-10-11 SC43   TSO       TOLOD    ACCNT#   +.173347666666667E+02
 2007-10-10 2007-10-11 SC47   TSO       RC47     ACCNT#   +.335626805555556E+02
 2007-10-10 2007-10-11 SC48   STC       A4SR01AA -        +.153323083333333E+02
 2007-10-10 2007-10-11 SC48   STC       A4SR01AS -        +.153321500000000E+02
 2007-10-10 2007-10-11 ZT01   JES2      LYSTE1CT 0        +.185042361111111E+02
 2007-10-10 2007-10-11 ZT01   STC       ED01BRK  -        +.903513611111111E+02
F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure 7-28 Display SAMPLE_ITUAM_V

7.3.2 Loading user-defined data

Figure 7-29 on page 237 shows the structure of the Job File that we will use to load user-defined Tivoli Decision Support for z/OS data in Tivoli Usage and Accounting Manager. You can create a template for this new file by copying the file we used in Figure 7-11 on page 221 and deleting all but one of the Integrator steps.

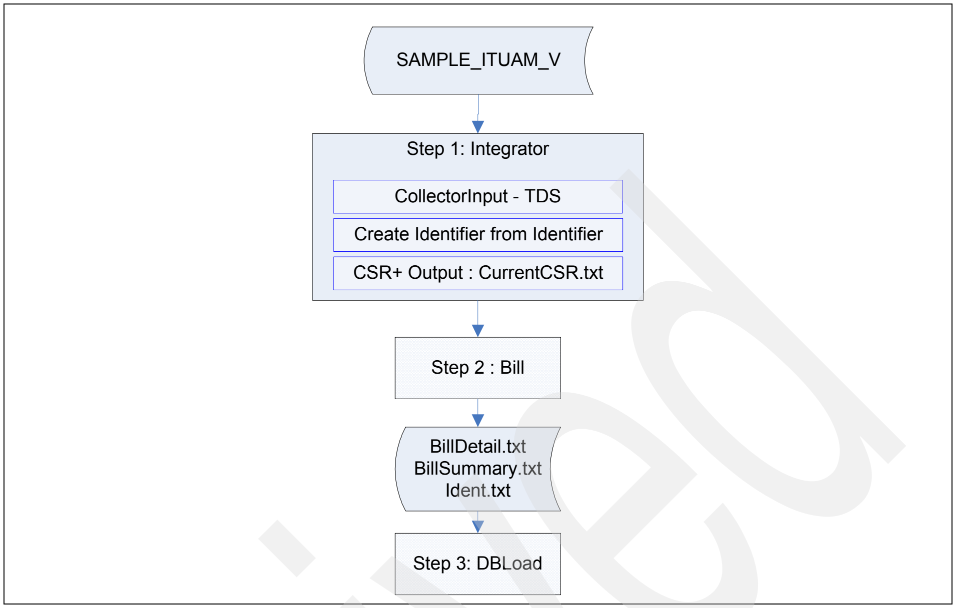


Figure 7-29 Sample user defined table processing

In this section we detail the steps for defining the job file. For a review of the job file structure, see Chapter 3, "Data collection and processing" on page 39. The complete listing of the job is provided in "Sample job for z/OS user defined data load" on page 340 in the appendix.

1. Modify the Job and process tags and specify the appropriate ID and descriptions. Most of the other parameters can use the defaults.
2. The input stage of the integrator step contains the following:
   - SQL for TDS collector that retrieves data from SAMPLE_ITUAM_V
   - CSR field mapping definition
3. The Collector part contains the SQL statement to be executed. It is a good idea to test this query in the Tivoli Decision Support for z/OS report dialog and make sure it works before you use it in here. The two question marks are replacements for the &FROM_DATE and &TO_DATE variables commonly used in Tivoli Decision Support for z/OS queries and supplied as parameters. The SQL statement that we use is shown in Example 7-11.

Example 7-11 SQL statement

```
<Collector name ="TDS">
<Connection dataSourceName="DB8D"/>
<Statement text="SELECT DATE_END AS DATE,
                 MVS_SYSTEM_ID SYSID,
```

```
                JOB_NAME AS JOBNAME,
                ACCOUNT_FIELD1 AS ACCOUNT,
                DECIMAL(SUM(ELAPSED_HOURS),20,6) AS ELAPSEDH  ,
                DECIMAL(SUM(CPU_SECONDS),20,6)   AS CPUSEC    ,
                DECIMAL(SUM(ZAAP_SECONDS),20,6)  AS ZAAPSEC   ,
                DECIMAL(SUM(ZIIP_SECONDS),20,6)  AS ZIIPSEC   ,
                DECIMAL(SUM(IO_SERVICE_UNITS))   AS IOSU      ,
                DECIMAL(SUM(JOB_COUNT)) AS JOBS
            FROM DRL.SAMPLE_ITUAM_V
            WHERE DATE_END BETWEEN ? AND ?
            GROUP BY
            DATE_END, MVS_SYSTEM_ID, JOB_NAME, ACCOUNT_FIELD1"/>
<Parameter src="PARAMETER" sqlType="DATE" position="1"
                srcName="StartLogDate"/>
<Parameter src="PARAMETER" sqlType="DATE" position="2"
                srcName="EndLogDate"/>
</Collector>
```

4. The parameters that we use are shown in Example 7-12. Here, we change the Resourceheader and Feed values.

Example 7-12 Parameters

```
<Parameter name="StartLogDate" value="%LogDate_Start%"
                 dataType="DATETIME" format="yyyyMMdd"/>
<Parameter name="EndLogDate" value="%LogDate_End%"
                 dataType="DATETIME" format="yyyyMMdd"/>
<Parameter name="Resourceheader" Value="TDSzUSER"
                 dataType="STRING"/>
<Parameter name="Feed" value="TDSzUSER"
                 dataType="STRING"/>
<Parameter name="LogDate" value="%LogDate_End%"
                 dataType="DATETIME" format="yyyyMMdd"/>
```

5. The data mapping for creating the CSR format is defined in the InputFields and OutputFields tags. The input fields are the labelling of the SQL SELECT statement output, while the output fields are CSR record definitions. The mapping is shown in Figure 7-30 on page 239.

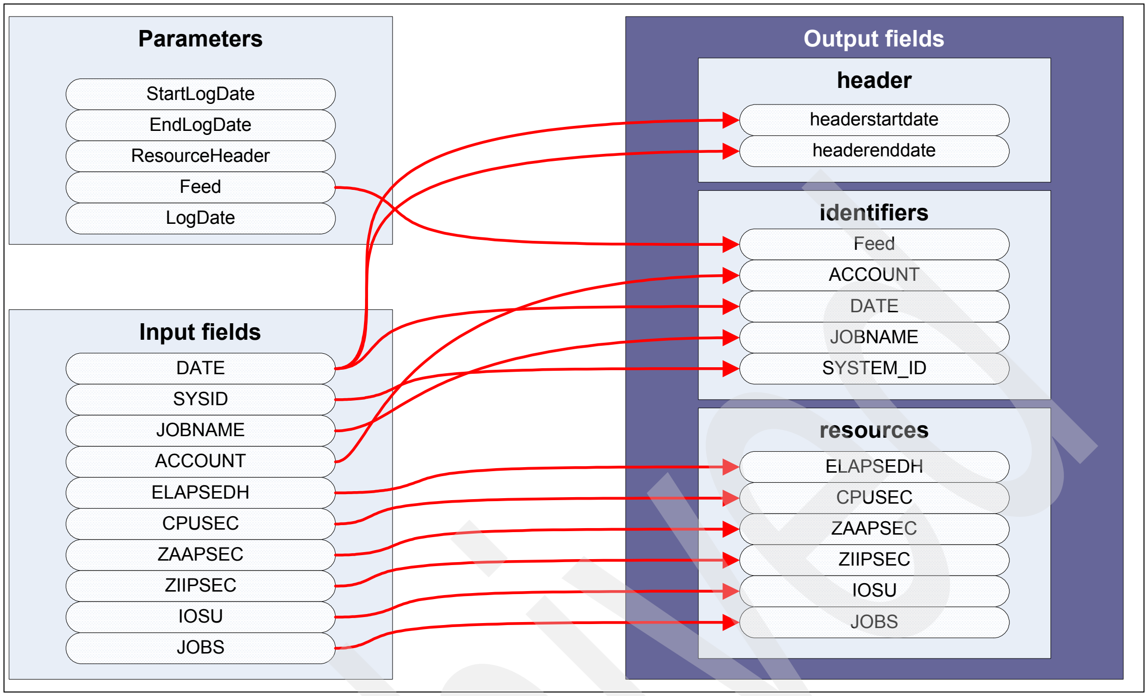


Figure 7-30 Field mapping

> **Note:** The output resources are metered resources that must be defined in the rate table for the Bill process to run without warning.

6. The CreateIdentifierFrom Identifiers stage that we have from the TDSzLoad1.xml file lets us define the Account_Code identifier based on the ACCOUNT identifier. Because we perform an account code translation in 7.3.4, "Account code translation example" on page 243, the content of this field can be arbitrary and will be replaced then. We keep the definition from TDSzLoad1.xml.
7. We specify the CSRoutput file to %ProcessFolder%/TDSzUSER/%LogDate_End%-TDSzUSER.txt
8. The rest of the job for Bill and DBLoad steps is similar to the TDSzLoad1.xml job. However, because we will still perform some translation for account code and CPU normalization later, we set `active="false"` for them just to verify our initial conversion.

Note: The scan step is not necessary because we only have a single integrator step output and do not need to perform a merging scan.

The Job File has now been created and we save it in the /opt/ibm/tuam/jobfiles with the name of LoadTDSzUser1.xml.

7.3.3 Defining rate codes

We now have to define the rate codes for the output resources. We add the missing Rates, but add them in a new Rate Group just to keep them separated as user-defined. Select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Rates** and click the **New** button. Add the IOSU Rate Code as shown in Figure 7-31 on page 241.

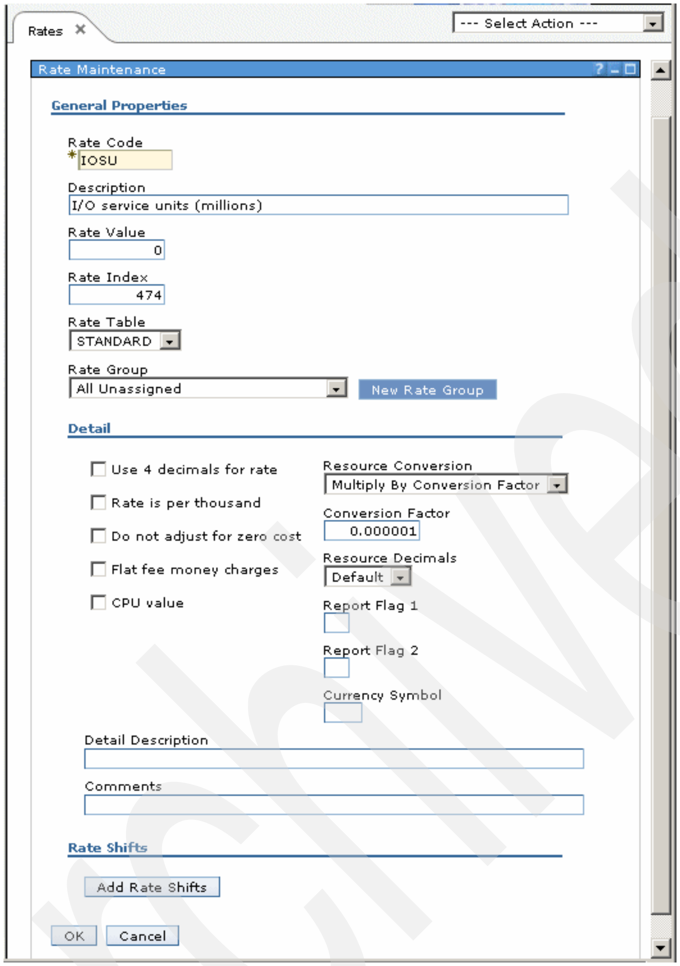


Figure 7-31 Adding the new IOSU Rate Code

Define the IOSU Rate Code as millions of I/O Service Units. For this, use the Resource Conversion and multiply the measure with a factor of 0.000001. Then click the **New Rate Group** button and a screen like the one in Figure 7-32 on page 242 will be displayed.

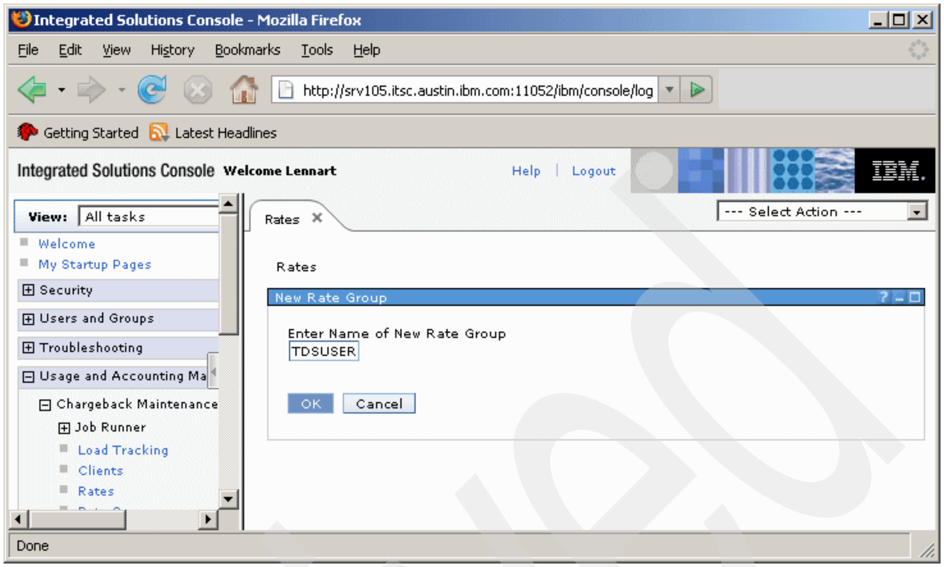


Figure 7-32 Add a new Rate Group

Type a name for the new Rate Group, for example `TDSUSER`, and click **OK**. That will bring you back to the screen in Figure 7-31, with the Rate Group box filled in with TDSUSER, where you can now click the **OK** button.

> **Note:** For the IOSU Rate Code we used a conversion factor of 0.000001 because the number of Service Units, SU, are usually large numbers. So by having pricing per one million SU we can put the price at, for example, $1 instead of $0.000001.

Continue with adding the other Rate Codes, without conversion factors. The Rate Group TDSUSER is now defined and will be found in the Rate Codes drop-down list in the screen shown in Figure 7-31.

Because no message is issued when we add a new Rate Group, it is good to check the contents of the new Rate Group. Do this by selecting **Usage and Accounting Manager** → **Chargeback Maintenance** → **Rate Groups** and expanding the group TDSUSER as in Figure 7-33 on page 243.

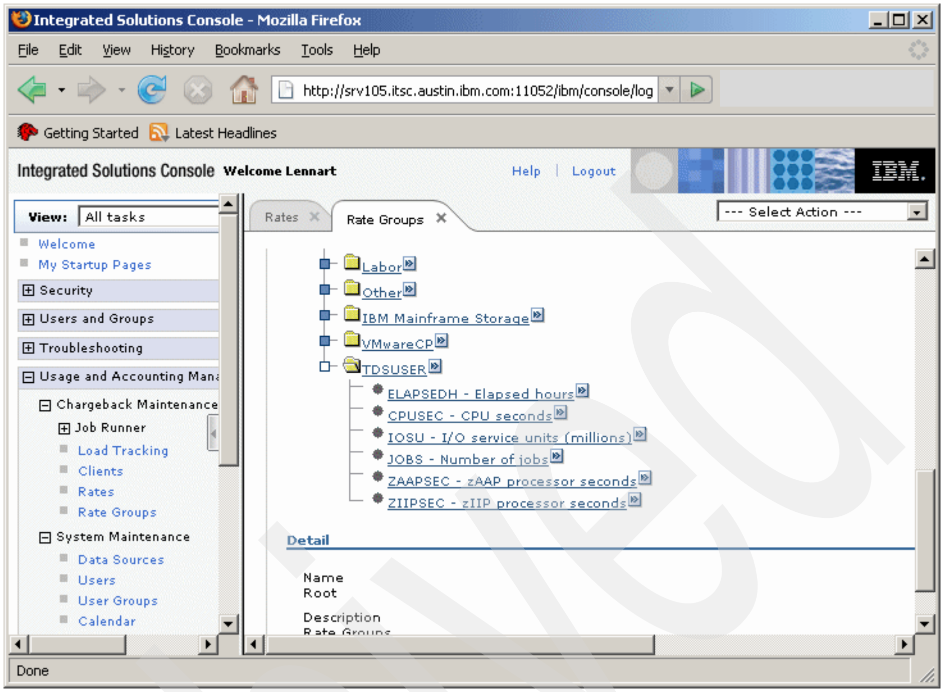


Figure 7-33 Rates in the TDSUSER Group

You are now ready to run the LoadTDSzUser1.xml Job File. Start the ISC and select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Job Runner** → **Job Files** and find the LoadTDSzUser1.xml job. Validate and run the job. The job should issue the message `The job completed successfully`.

So far we have created CSR records from user-defined Tivoli Decision Support for z/OS data. The data was not loaded in Tivoli Usage and Accounting Manager because we coded `active="false"` in the DatabaseLoad step.

7.3.4 Account code translation example

Your user-defined Tivoli Decision Support for z/OS data may already have a correct accounting code. If so, you can use that for the Tivoli Usage and Accounting Manager identifier Account_Code. It is more likely you will have to derive the accounting code from one or more columns. This can be done in Tivoli Decision Support for z/OS during collection by use of lookup tables. The account codes in the resource accounting tables are derived in this way, as we saw, for example, in “The RAFASTC lookup table” on page 210. If you want to use this method, the Tivoli Decision Support for z/OS UPDATE definition RAFSTC_UP can be used as a template. Another way of deriving the account code is to use SQL to join data from two or more tables.

In this example we assume an account code translation is needed and we will use some of the functions in Tivoli Usage and Accounting Manager. In our sample user-defined data, which is the SAMPLE_ITUAM_V view displayed in Figure 7-28 on page 236, we use the columns ACCOUNT_FIELD1 and JOBNAME to derive an account code for Tivoli Usage and Accounting Manager according to the following rules:

- If the length of the ACOOUNT_FIELD1 is at least four characters, then take the first four characters as the account code.
- If the length of the ACCOUNT_FIELD1 is less than four characters, take the first character in the JOBNAME and do an Identifier Conversion returning a four character account code.

To implement this we need a number of stages in the Integrator step. Start by making a copy of the LoadTDSzUser1.xml file; in our example, we called it LoadTDSzUser2.xml. The LoadTDSzUser1.xml has two stages in the Integrator step: CreateIdentifierFromIdentifier and CSROutput. We added several more stages. The new stages are:

1. CreateIdentifierFromValue. This stage will create a new identifier, Temp1.
2. IdentifierConversionFromTable. This stage will convert the first character in the JOBNAME to a four character account code and store it in the Temp1 identifier.
3. CreateIdentifierFromIdentifiers. This stage will concatenate the Temp1 identifier and the ACCOUNT identifier, which is the ACCOUNT_FIELD1 column from our SAMPLE_ITUAM_V view, and store it the Temp2 identifier.
4. CreateIdentifierFromRegEx. This stage will use a regular expression to extract the first four character word in the Temp2 identifier and store it in the Account_Code identifier. For information about regular expression, see:

   `http://en.wikipedia.org/wiki/Regular_expression`
5. DropFields. This stage will drop Temp1 and Temp2.
6. CSROutput. This stage will produce the CSR file.

The stages in the integrator step will look like Example 7-13.

Example 7-13 Stages for account translation

```
<Stage name="CreateIdentifierFromValue" active="true">
    <Identifiers>
    <Identifier name="Temp1" value="?"/>
    </Identifiers>
```

```
    <Parameters>
    <Parameter modifyIfExists="true"/>
    </Parameters>
</Stage>

<Stage name="IdentifierConversionFromTable" active="true">
  <Identifiers>
  <Identifier name="Temp1">
  <FromIdentifiers>
  <FromIdentifier name="JOBNAME" offset="1" length="1"/>
  </FromIdentifiers>
  </Identifier>
  </Identifiers>
  <Files>
  <File name="/opt/ibm/tuam/processes/AccttablTDSzUser.txt"
              type="table"/>
  <File name="Exception.txt" type="exception" format="CSROutput"/>
  </Files>
  <Parameters>
  <Parameter exceptionProcess="true"/>
  <Parameter sort="true"/>
  <Parameter upperCase="false"/>
  <Parameter writeNoMatch="false"/>
  </Parameters>
</Stage>

<Stage name="CreateIdentifierFromIdentifiers" active="true">
  <Identifiers>
  <Identifier name="Temp2">
  <FromIdentifiers>
  <FromIdentifier name="Temp1" offset="1" length="4" delimiter=" " />
  <FromIdentifier name="ACCOUNT" offset="1" length="4" delimiter=" "/>
  </FromIdentifiers>
  </Identifier>
  </Identifiers>
  <Parameters>
  <Parameter keepLength="false"/>
  <Parameter modifyIfExists="true"/>
  </Parameters>
</Stage>

<Stage name="CreateIdentifierFromRegEx" active="true" trace="false" >
  <Identifiers>
  <Identifier name="Account_Code">
  <FromIdentifiers>
```

```
  <FromIdentifier name="Temp2" regEx=".*(\w{4}).*" value="$1"/>
  </FromIdentifiers>
  </Identifier>
  </Identifiers>
  <Parameters>
  <Parameter modifyIfExists="true"/>
  </Parameters>
</Stage>

<Stage name="DropFields" active="true">
  <Fields>
  <Field name="Temp1"/>
  <Field name="Temp2"/>
  </Fields>
</Stage>

<Stage name="CSROutput" active="true">
  <Files>
  <File name="%ProcessFolder%/TDSzUSER/%LogDate_End%-TDSzUSER.txt"  />
  </Files>
</Stage>
```

The second stage, IdentifierConversionFromTable, requires an account translation table. This is a text file; the file we used is in Example 7-14. This file must exist before you attempt to run the new Job File, LoadTDSzUser2.xml.

Example 7-14 Account translation table AccttablTDSzUser.txt

```
A,,AAAA
B,,BBBB
C,,CCCC
D,,DDDD
E,,EEEE
F,,FFFF
G,K,GGKK
L,P,LLPP
Q,S,QQSS
T,V,TTVV
W,Z,WWZZ
```

The Job File LoadTDSzUser2.xml is ready to run, without the last two steps, DatabaseLoad and Cleanup, because `active="false"` is coded in both of them. We run this new Job File.

We have still not loaded any user-defined Tivoli Decision Support for z/OS data into Tivoli Usage and Accounting Manager, and only produced CSR records. Let us have a look at them. Example 7-15 shows three CSR records from the /opt/ibm/tuam/processes/IBMTDSUSER/CurrentCSR.txt file created in the run of the LoadTDSzUser2.xml Job File.

Example 7-15 CSR records from user-defined Tivoli Decision Support for z/OS data

```
TDSzUSER,20071015,20071015,,,1,5,Feed,"TDSzUSER",DATE,"2007-10-15",SYSI
D,"SC47",JOBNAME,"COB1CTG
",Account_Code,"CCCC",5,ELAPSEDH,1096.436475,CPUSEC,23.400000,ZAAPSEC,7
69.780000,IOSU,5850,JOBS,1

TDSzUSER,20071015,20071015,,,1,6,Feed,"TDSzUSER",ACCOUNT,"ACCNT#",DATE,
"2007-10-15",SYSID,"SC47",JOBNAME,"DAVIS
",Account_Code,"ACCN",4,ELAPSEDH,1.662522,CPUSEC,0.620000,IOSU,508,JOBS
,1

TDSzUSER,20071015,20071015,,,1,6,Feed,"TDSzUSER",ACCOUNT,"999",DATE,"20
07-10-15",SYSID,"SC47",JOBNAME,"DB8CEJ1
",Account_Code,"DDDD",4,ELAPSEDH,0.002775,CPUSEC,0.590000,IOSU,1275,JOB
S,2
```

7.3.5 CPU normalization example

Your user-defined Tivoli Decision Support for z/OS data may already have normalized CPU values, in which case you don't have to use the Tivoli Usage and Accounting Manager functions to perform this task. The CPU data in SAMPLE_ITUAM_V is not normalized. In Example 7-1 on page 209 we populated the CPU_NORAMAL_DATA lookup table used in the Tivoli Decision Support for z/OS resource accounting component to do CPU normalization. In Tivoli Usage and Accounting Manager we can do the same thing, but in a different way; in this section we illustrate this for one system ID, SC47, and one identifier, CPUSEC. In ISC, select **Usage and Accounting Manager** → **System maintenance** → **CPU Normalization**. A screen like Figure 7-34 on page 248 is displayed.

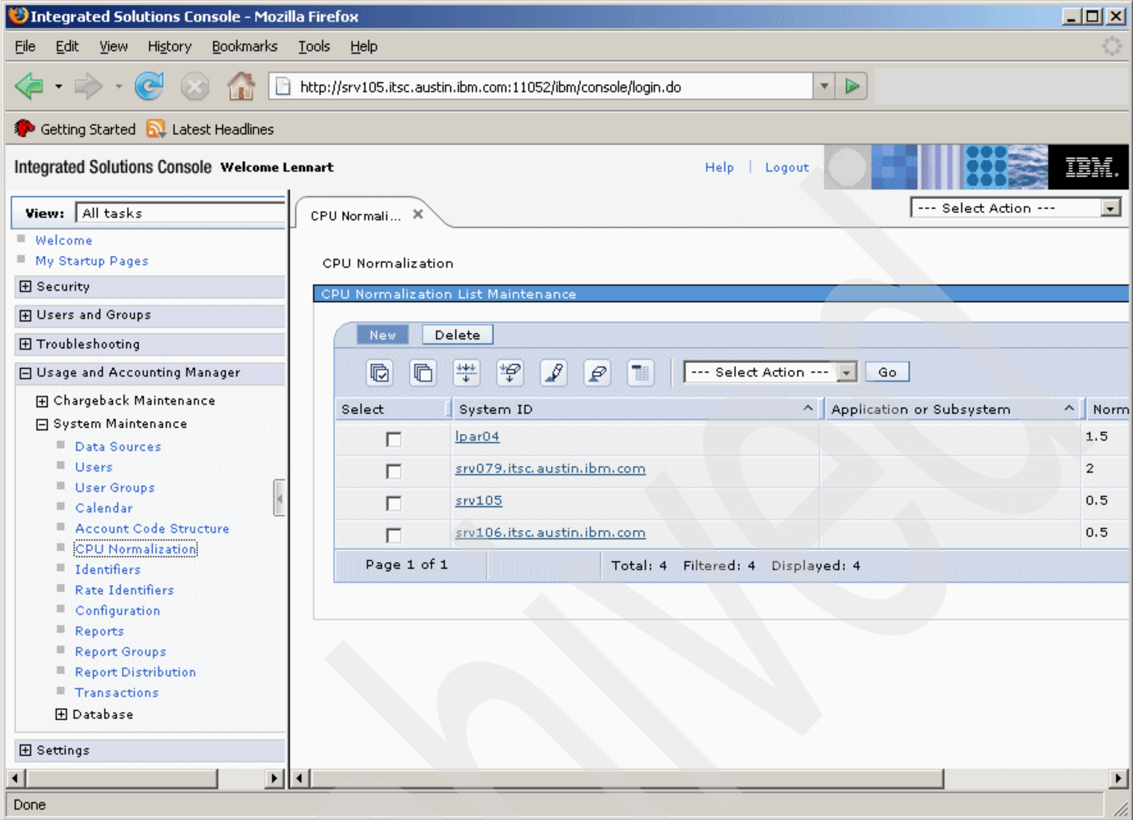


Figure 7-34 CPU Normalization list

Click the **New** button and fill in the fields as Figure 7-35 on page 249.

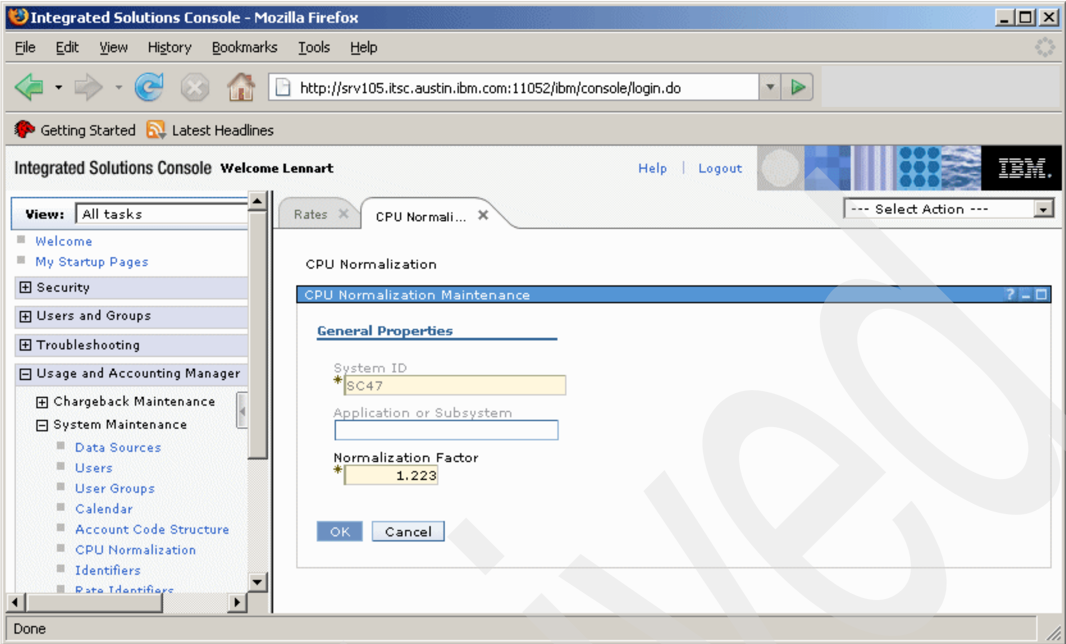


Figure 7-35 CPU Normalization new System ID

Leave the Application or Subsystem field blank. For Tivoli Decision Support for z/OS data this field is used to look at subsystem IDs. When we leave it blank the normalization factor applies to all subsystems for this system ID. Click the **OK** button.

> **Note:** The Normalization Factor is used as a multiplier for CPU identifiers. This is the inverse function of the one used for the Tivoli Decision Support for z/OS resource accounting tables. Here we use the factor 1.223 which is 1 / 0.8177.

The Rate Code CPUSEC must be defined as a CPU value for the normalization to take place. Select **Usage and Accounting Manager** → **Chargeback Maintenance** → **Rates** and filter out the CPUSEC Rate Code as shown in Figure 7-36 on page 250.

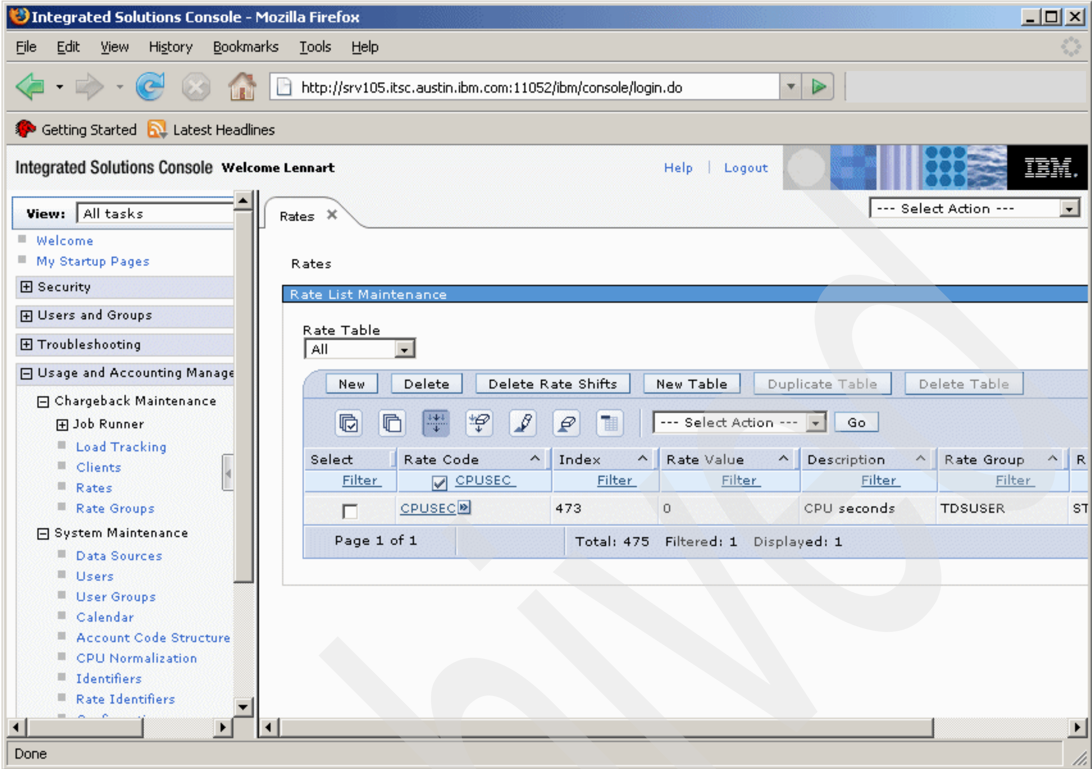


Figure 7-36 CPUSEC rate

Click the CPUSEC Rate Code and the details will be displayed as in Figure 7-37 on page 251.

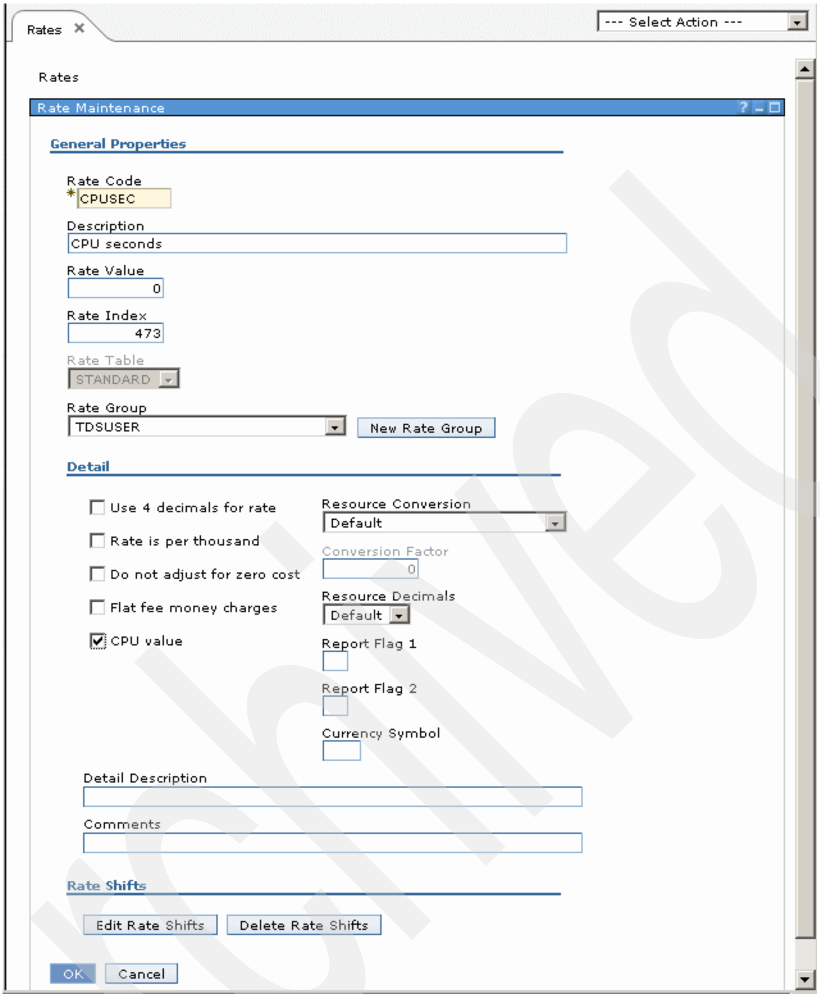


Figure 7-37 CPUSEC Rate details

Check the CPU value box and click the **OK** button. We have now set up Tivoli Usage and Accounting Manager to multiply CPUSEC by 1.223 for system ID SC47.

It is now time for the final run of the Job File LoadTDSzUser2.xml. We will do CPU normalization and execute all steps, including Database load. Edit the LoadTDSzUser2.xml and make the following changes:

- Change `active="false"` to `active="true"` for the rest of the steps.

- In the Billing step (`id="Process"`) add a Bill parameter:

  `<Parameter controlCard=”NORMALIZE CPU VALUES”/>`

  This parameter is required for Tivoli Usage and Accounting Manager to do normalization.

After you run the job, the result of CPU normalization cannot be seen in the CSR records. Instead we have to look in the BillDetail.txt file to verify that our CPU normalization worked as expected. Example 7-16 shows detail records corresponding to the CSR records in Example 7-15 on page 247.

Example 7-16 Detail records

```
991,TDSzUSER,200411,253,31,1,,SC47,,20071015,20071015,,,20071015,200710
15,1,2,"CCCC",,,5,ELAPSEDH,1096.436475,CPUSEC,28.618200000,ZAAPSEC,769.
780000,IOSU,5850,JOBS,1

991,TDSzUSER,200411,253,32,1,,SC47,,20071015,20071015,,,20071015,200710
15,1,2,"ACCN",,,4,ELAPSEDH,1.662522,CPUSEC,0.758260000,IOSU,508,JOBS,1

991,TDSzUSER,200411,253,33,1,,SC47,,20071015,20071015,,,20071015,200710
15,1,2,"DDDD",,,4,ELAPSEDH,0.917241,CPUSEC,1.137390000,IOSU,4797,JOBS,1
```

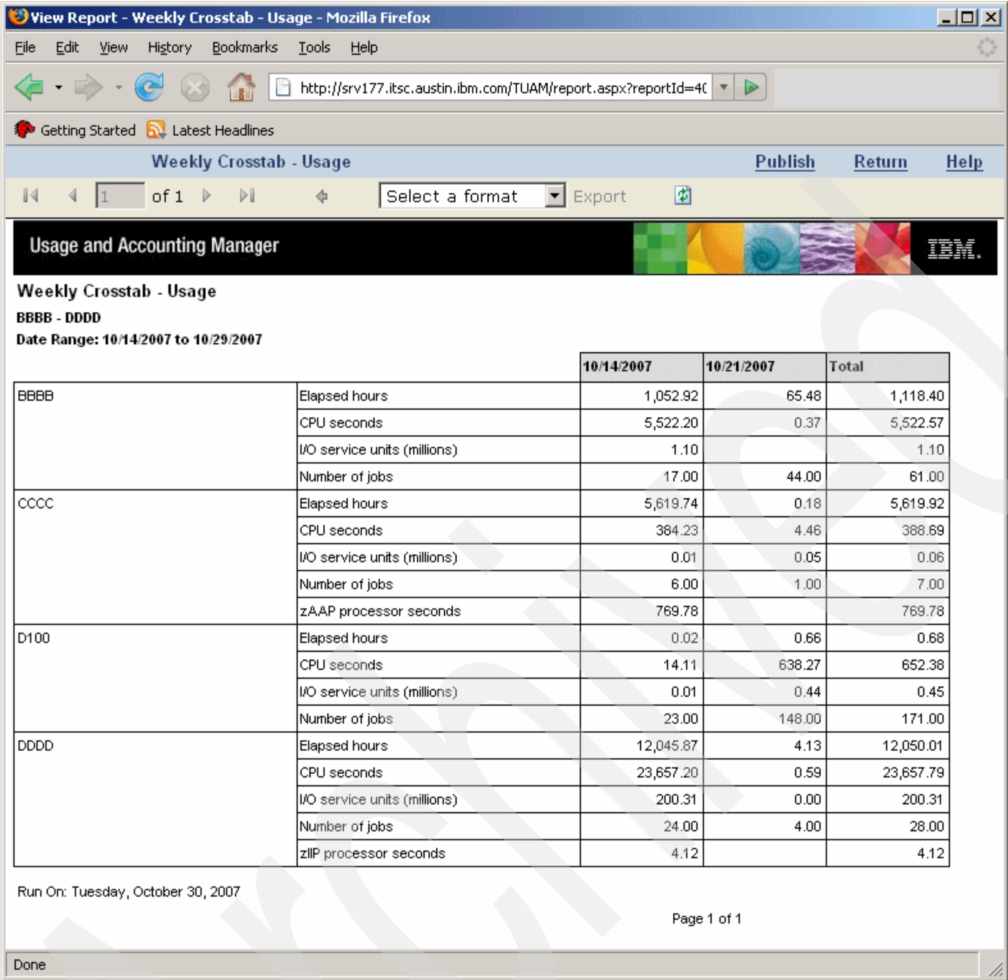
7.4 Sample reports

The next four figures show sample reports regarding data from Tivoli Decision Support for z/OS. The reports are accessed using the Web reporting feature from the Microsoft Internet Information Server. Our report server is located in:

`http://srv177/tuam`

The reports are:

- Weekly crosstab usage for accounts BBBB to DDDD, shown in Figure 7-38 on page 253.
- Summary crosstab 2 for usage accounting for mainframe charges, in Figure 7-39 on page 254.
- Summary crosstab 2 for usage accounting for user-defined table, in Figure 7-40 on page 255. Note that we have data for zAAP and zIIP processor usage.
- The detail for user-defined data, in Figure 7-41 on page 256



Weekly Crosstab - Usage

BBBB - DDDD

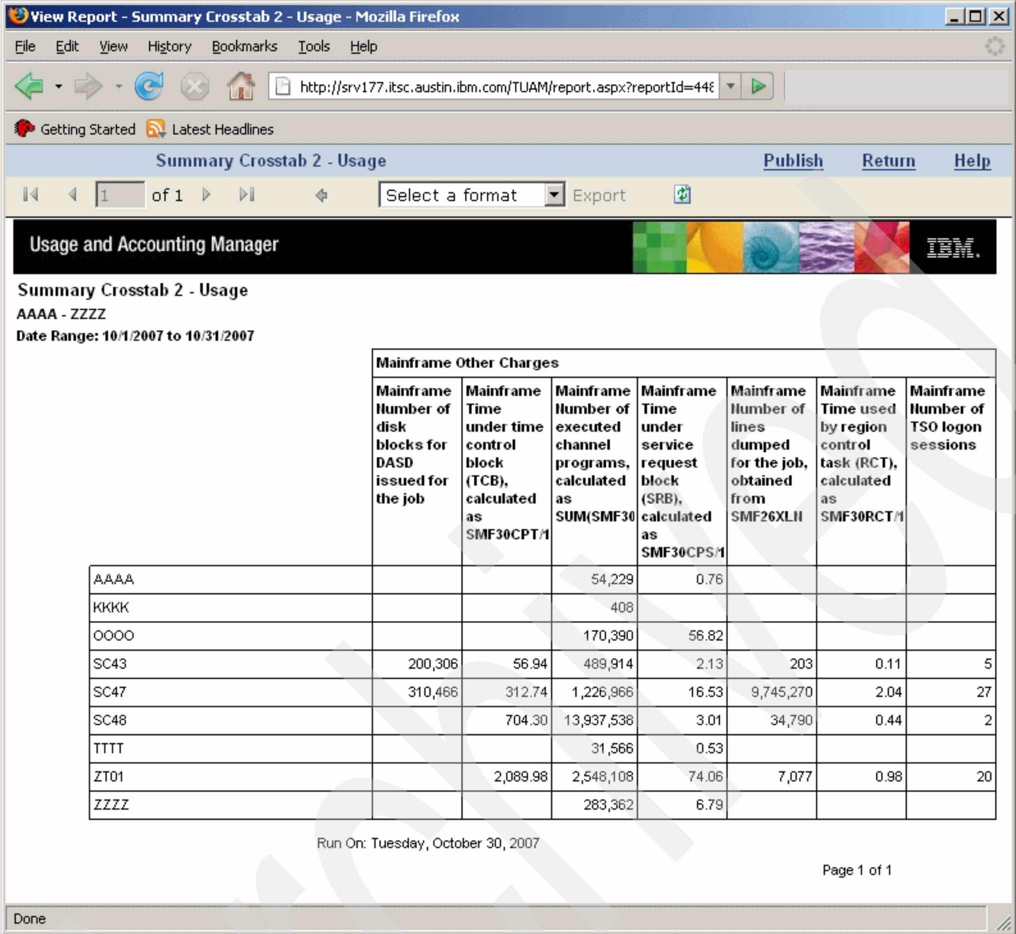
Date Range: 10/14/2007 to 10/29/2007

		10/14/2007	10/21/2007	Total
BBBB	Elapsed hours	1,052.92	65.48	1,118.40
	CPU seconds	5,522.20	0.37	5,522.57
	I/O service units (millions)	1.10		1.10
	Number of jobs	17.00	44.00	61.00
CCCC	Elapsed hours	5,619.74	0.18	5,619.92
	CPU seconds	384.23	4.46	388.69
	I/O service units (millions)	0.01	0.05	0.06
	Number of jobs	6.00	1.00	7.00
	zAAP processor seconds	769.78		769.78
D100	Elapsed hours	0.02	0.66	0.68
	CPU seconds	14.11	638.27	652.38
	I/O service units (millions)	0.01	0.44	0.45
	Number of jobs	23.00	148.00	171.00
DDDD	Elapsed hours	12,045.87	4.13	12,050.01
	CPU seconds	23,657.20	0.59	23,657.79
	I/O service units (millions)	200.31	0.00	200.31
	Number of jobs	24.00	4.00	28.00
	zIIP processor seconds	4.12		4.12

Run On: Tuesday, October 30, 2007

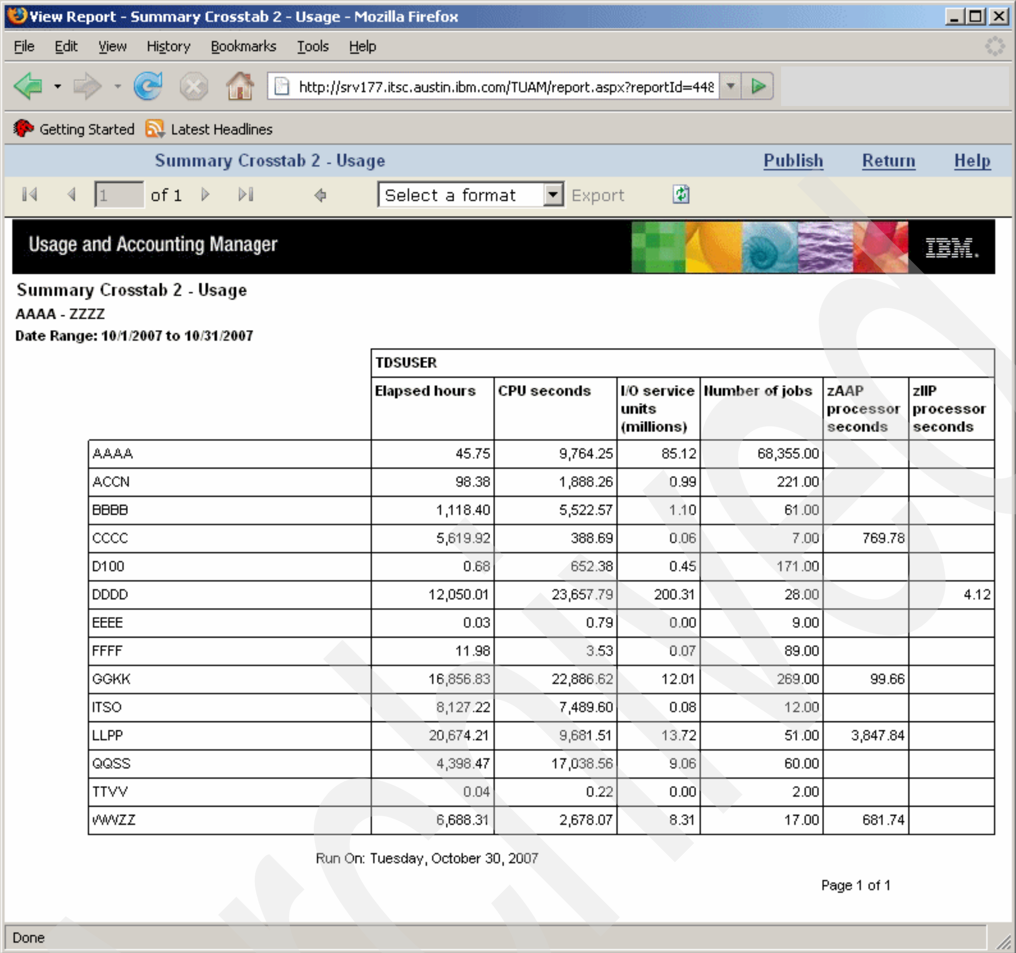
Page 1 of 1

Figure 7-38 Sample report for weekly usage data z/OS



	Mainframe Other Charges						
	Mainframe Number of disk blocks for DASD issued for the job	Mainframe Time under time control block (TCB), calculated as SMF30CPT/1	Mainframe Number of executed channel programs, calculated as SUM(SMF30	Mainframe Time under service request block (SRB), calculated as SMF30CPS/1	Mainframe Number of lines dumped for the job, obtained from SMF26XLN	Mainframe Time used by region control task (RCT), calculated as SMF30RCT/1	Mainframe Number of TSO logon sessions
AAAA			54,229	0.76			
KKKK			408				
OOOO			170,390	56.82			
SC43	200,306	56.94	489,914	2.13	203	0.11	5
SC47	310,466	312.74	1,226,966	16.53	9,745,270	2.04	27
SC48		704.30	13,937,538	3.01	34,790	0.44	2
TTTT			31,566	0.53			
ZT01		2,089.98	2,548,108	74.06	7,077	0.98	20
ZZZZ			283,362	6.79			

Figure 7-39 Sample report with standard z/OS accounting data



Summary Crosstab 2 - Usage
AAAA - ZZZZ
Date Range: 10/1/2007 to 10/31/2007

	TDSUSER					
	Elapsed hours	CPU seconds	I/O service units (millions)	Number of jobs	zAAP processor seconds	zIIP processor seconds
AAAA	45.75	9,764.25	85.12	68,355.00		
ACCN	98.38	1,888.26	0.99	221.00		
BBBB	1,118.40	5,522.57	1.10	61.00		
CCCC	5,619.92	388.69	0.06	7.00	769.78	
D100	0.68	652.38	0.45	171.00		
DDDD	12,050.01	23,657.79	200.31	28.00		4.12
EEEE	0.03	0.79	0.00	9.00		
FFFF	11.98	3.53	0.07	89.00		
GGKK	16,856.83	22,886.62	12.01	269.00	99.66	
ITSO	8,127.22	7,489.60	0.08	12.00		
LLPP	20,674.21	9,681.51	13.72	51.00	3,847.84	
QQSS	4,398.47	17,038.56	9.06	60.00		
TTVV	0.04	0.22	0.00	2.00		
WWZZ	6,688.31	2,678.07	8.31	17.00	681.74	

Run On: Tuesday, October 30, 2007

Page 1 of 1

Figure 7-40 Sample report for usage of user defined z/OS data

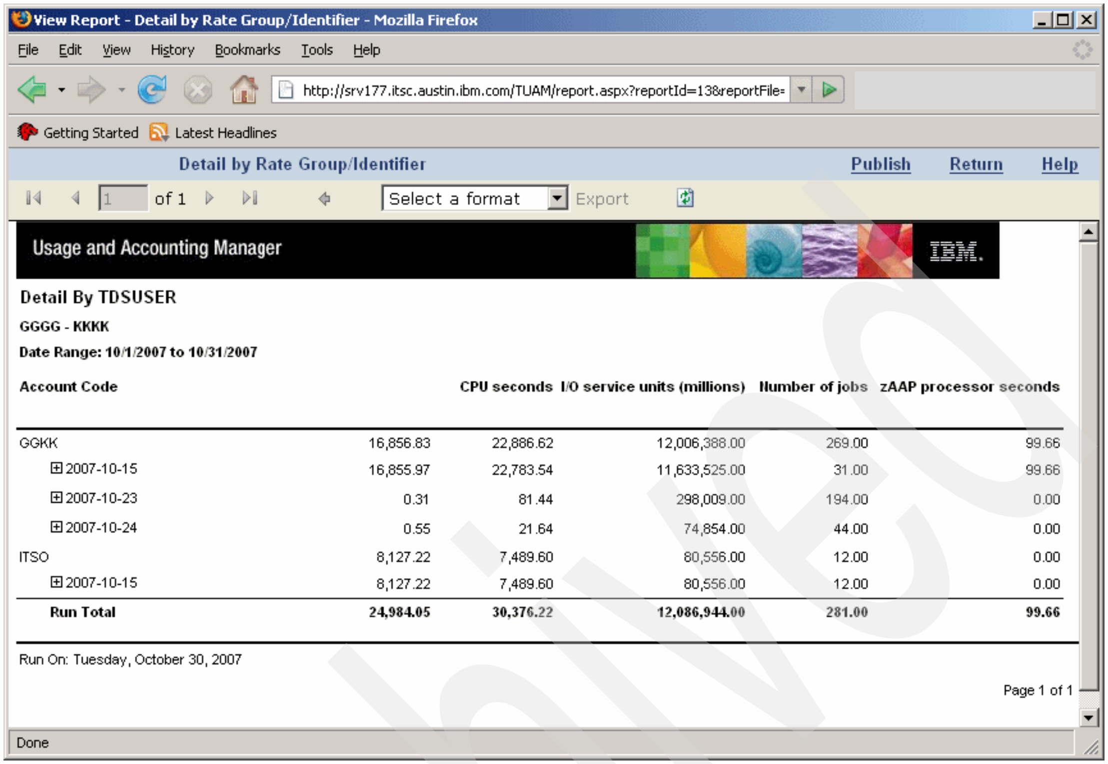


Figure 7-41 Sample report for detail by TDSUSER data

8

Financial Modeler, non-usage transactions and database size calculation

This chapter provides an overview of some add-on components for Tivoli Usage and Accounting Manager. The components covered are:

8.1 Financial Modeler

The Financial Modeler is a Web-based spreadsheet application that is supplied as is from Tivoli Usage and Accounting Manager. It allows the calculation of IT budget allocation and rates based on the usage data collected by Tivoli Usage and Accounting Manager, and It assists you in assigning rates for the rate codes. Before you can use the tool you must first have usage data collected over a period of time. You can start using the Financial Modeler with little data and re-run the modeling any time when you have collected more.

The Financial Modeler is installed within Microsoft Internet Information Server. It is typically installed together with the Web reporting server. To use the Financial Modeler, you must use Microsoft Internet Explorer®.

A conceptual overview of the Financial Modeler is shown in Figure 8-1.

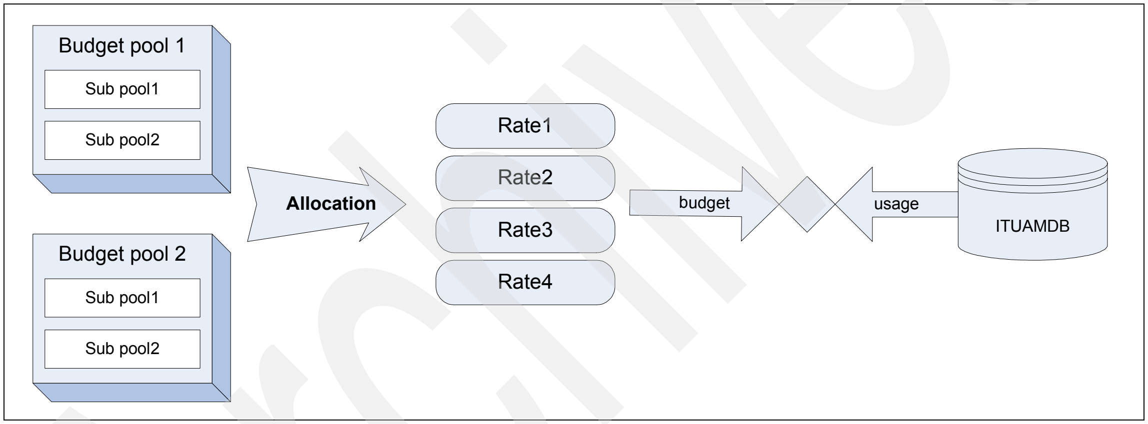


Figure 8-1 Financial modeler concept

A model compares budget and usage information. The budget consists of a set of budget pools and their sub pools. You define the budget pools for money amounts and allocate the monies to the sub-pools. The budget is then allocated (by percentage) to specific usage rate codes. This allows the spreadsheet to calculate the monthly budget money available per rate code.

The Financial Modeler then allows data to be retrieved from the Tivoli Usage and Accounting Manager usage database. The usage data is extrapolated against the calculated budget to demonstrate a break-even rate for the budget. You can then play around to perform allocation changes, uplift of cost and other analysis. The results can be fed back to the Tivoli Usage and Accounting Manager rate table.

You can have multiple models that define your different budget allocation areas. These models are saved as XML files within the directory %TUAM_HOME%\server\financial_modeler\data.

To use the Financial Modeler your user ID must belong to a group that has the Group Privilege Allow Financial Modeler access checked, as shown in Figure 8-2. Open the ISC menu and select **Usage and Accounting Manager** → **System Maintenance** → **User Groups**. Click the “**>>**” button next to your group. Select **Edit** from the submenu to check and modify the settings.

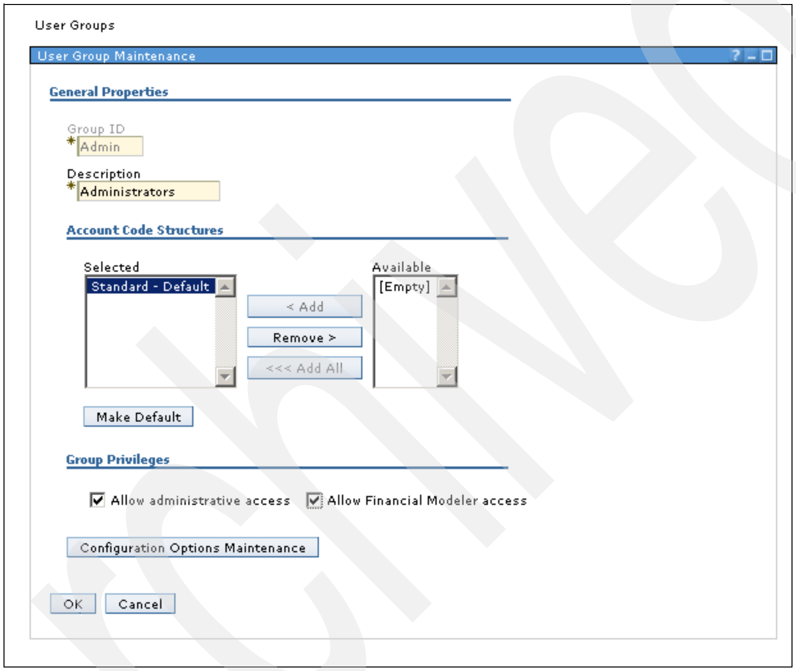


Figure 8-2 User group maintenance

We walk through the steps for using the Financial Modeler in this section. In this example, we are working with:

- A budget system that has 2 pools, with 200,000 for mainframe maintenance and 250,000 for the distributed system.
- We are analyzing z/OS, Windows, and UNIX server rates and we assume CPU usage is the chargeback criteria.

1. Log in to the Financial Modeler (Figure 8-3 on page 260).

   a. The URL for our Financial Modeler is:
   `http://srv177/FinancialModeler`

b. Log in with the user and password that has access to Financial Modeler. We used the admin user.

c. Click **Cancel** when prompted for opening a model because we will create a new model.

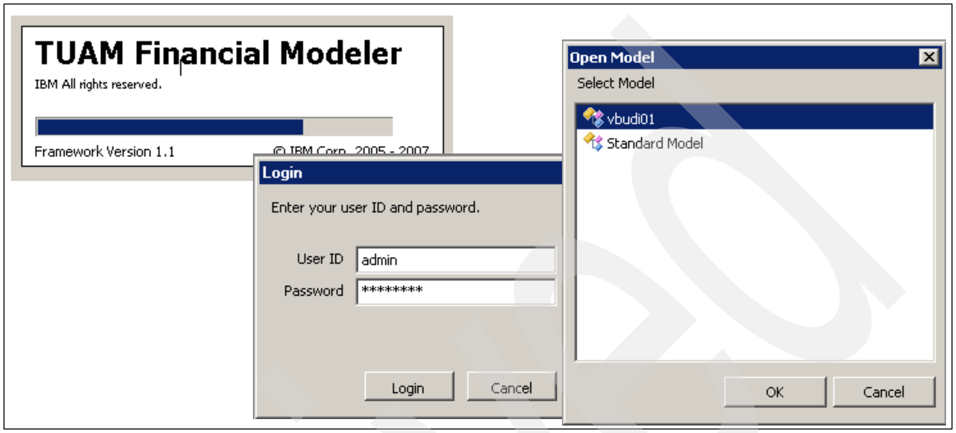


Figure 8-3 Starting Financial Modeler

2. A new model wizard is created when you click the **New** button. The wizard collects information about:

   – Budget pools
   – Budget sub-pools
   – Rate codes

   The panels the wizard takes you through are shown in Figure 8-4 on page 261.

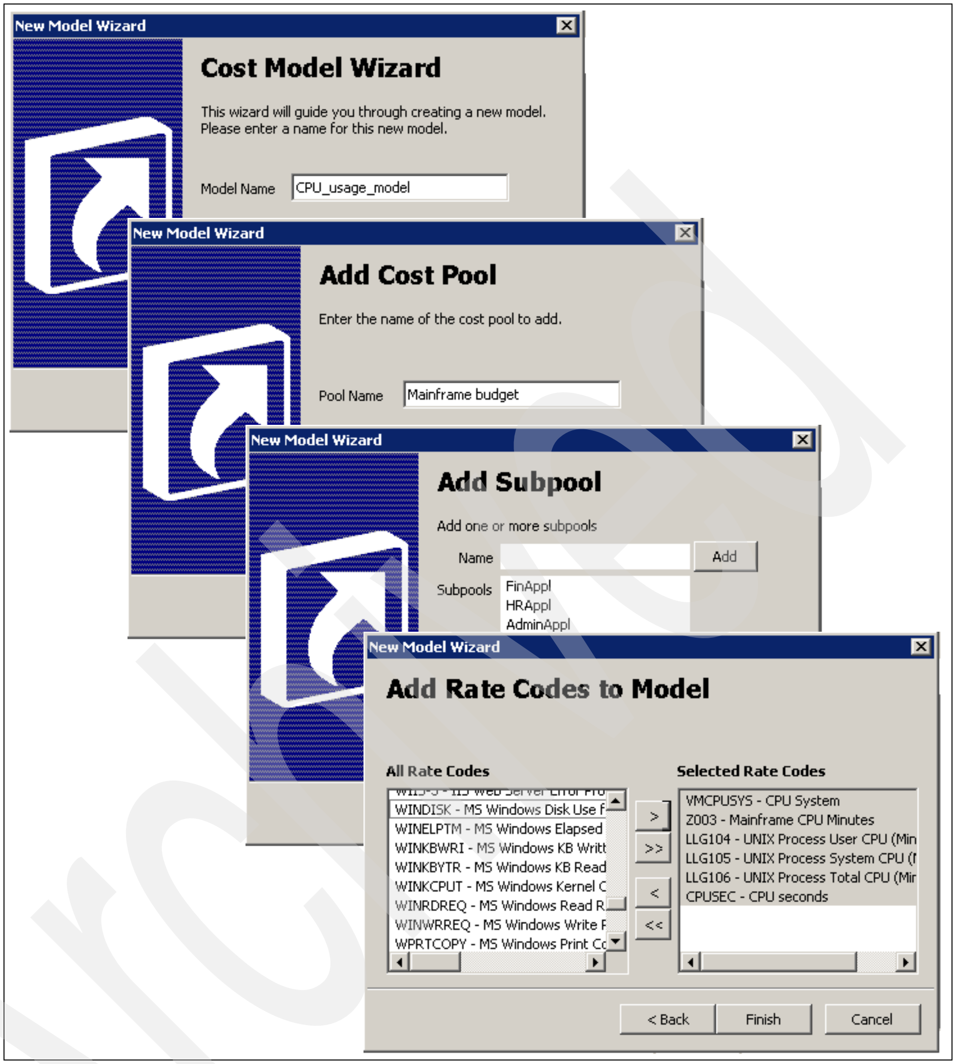


Figure 8-4 Model creation wizard

3. Once the model is created, we get the spreadsheet view. The view has four tabs: Budget Values, Percent Allocation, Cost Calculations, and Rate Calculations. Examples of these spreadsheets follow.

– Budget Values allows you to enter, view, and change budget numbers (Figure 8-5).

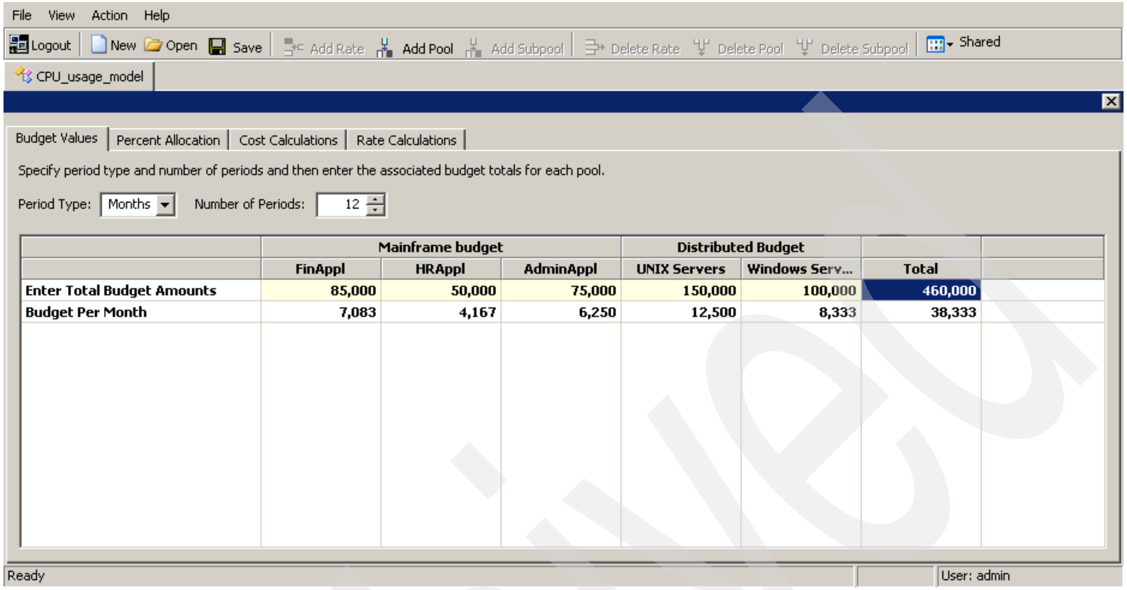


Figure 8-5 Budget Values

– The Percent Allocation tab (Figure 8-6) is where you assign percentage values for the rate groups from the budget sub pools.

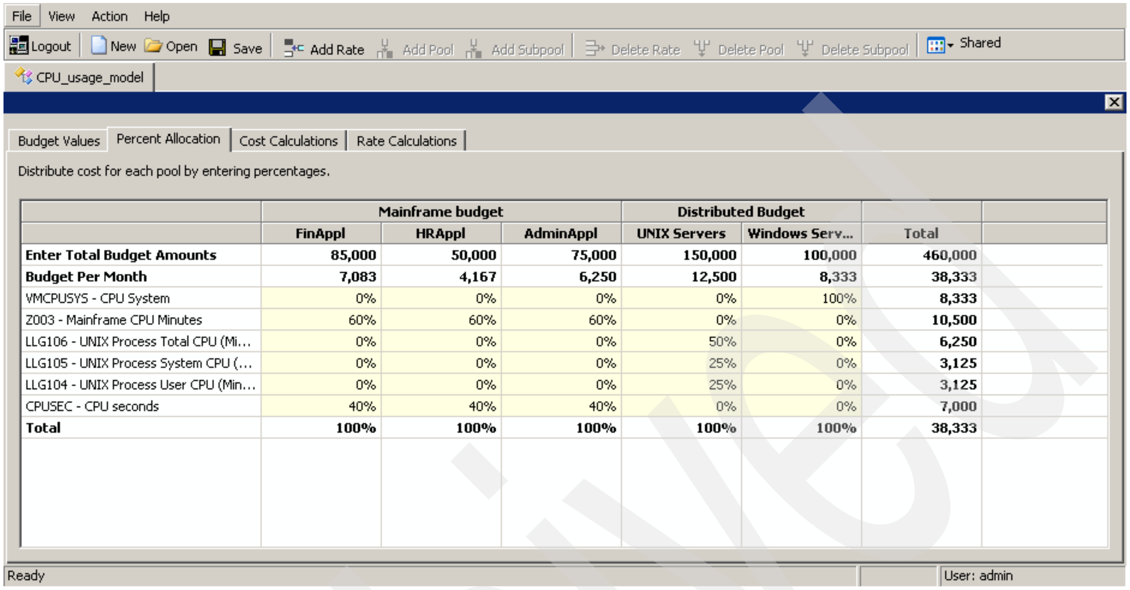
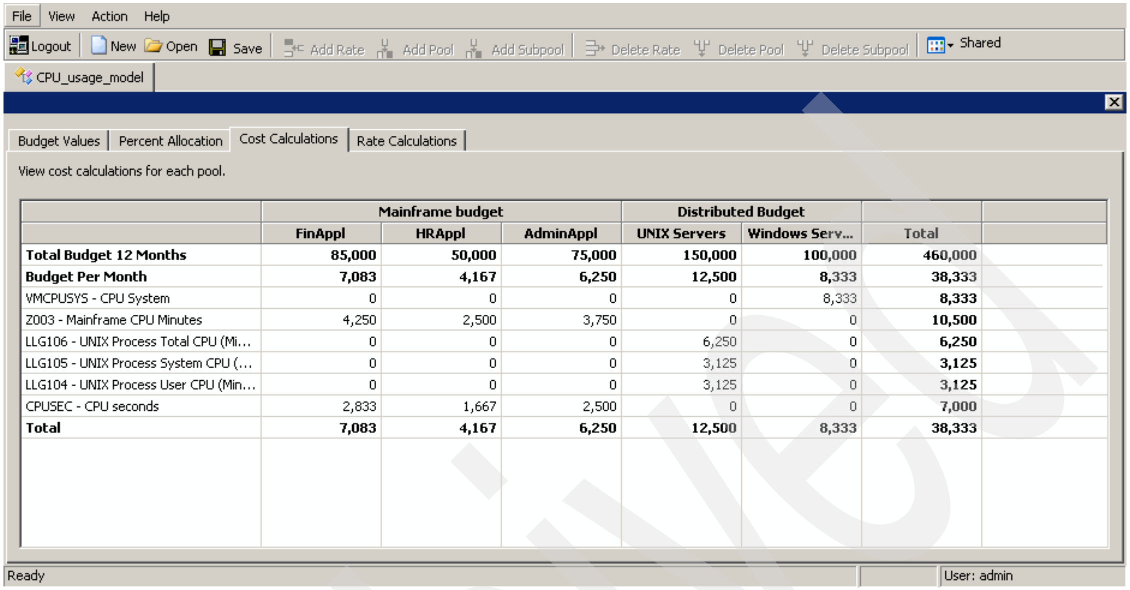


Figure 8-6 Percent Allocation

> **Note:** A subpool relates to one or more Rate Codes. You can choose any percentages you want; but they must add up to 100%.

– The Cost Calculations spreadsheet uses the allocation information to calculate the cost for the period for each resource (Figure 8-7).



	Mainframe budget			Distributed Budget			
	FinAppl	HRAppl	AdminAppl	UNIX Servers	Windows Serv...	Total	
Total Budget 12 Months	**85,000**	**50,000**	**75,000**	**150,000**	**100,000**	**460,000**	
Budget Per Month	**7,083**	**4,167**	**6,250**	**12,500**	**8,333**	**38,333**	
VMCPUSYS - CPU System	0	0	0	0	8,333	**8,333**	
Z003 - Mainframe CPU Minutes	4,250	2,500	3,750	0	0	**10,500**	
LLG106 - UNIX Process Total CPU (Mi...	0	0	0	6,250	0	**6,250**	
LLG105 - UNIX Process System CPU (...	0	0	0	3,125	0	**3,125**	
LLG104 - UNIX Process User CPU (Min...	0	0	0	3,125	0	**3,125**	
CPUSEC - CPU seconds	2,833	1,667	2,500	0	0	**7,000**	
Total	**7,083**	**4,167**	**6,250**	**12,500**	**8,333**	**38,333**	

Figure 8-7 Cost calculation

– On the Rate Calculations page the model takes usage data retrieved from the database and uses it, along with budget allocation numbers, to calculate rates (Figure 8-8). You can refresh the values using the **Calculate Rates** button after changing the date selection. The computed values are shown in the yellow shaded column on the right side of the sheet.

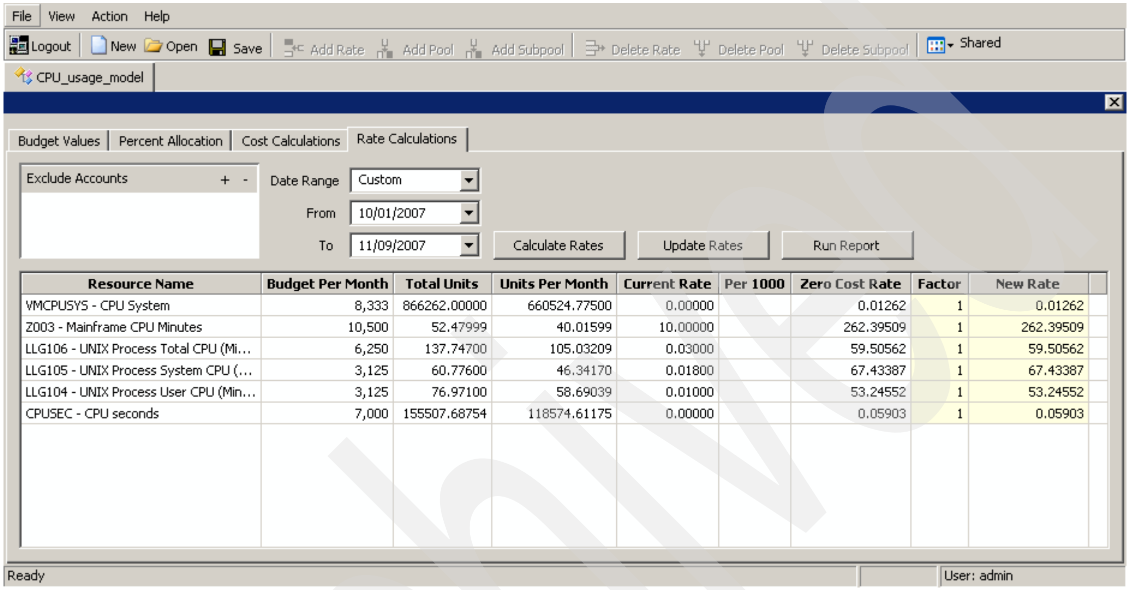


Figure 8-8 Rate calculation

Notes:

- The default rate calculation calculates for a zero profit.
- You can adjust the time period that you retrieve the data from.
- You can change the uplift Factor to adjust the rate.
- Click on **Update Rates** to save the calculated rates.

4. To save the model, click the **Save** button.

8.2 Transaction data collector

To address non-usage-based billing within the IT environment, Tivoli Usage and Accounting Manager provides a table for this kind of transaction.

The transaction job is designed for monthly usage only, so all transactions will be added at one time by running the job each month, depending on the organization. Figure 8-9 is an overview of the transaction function.

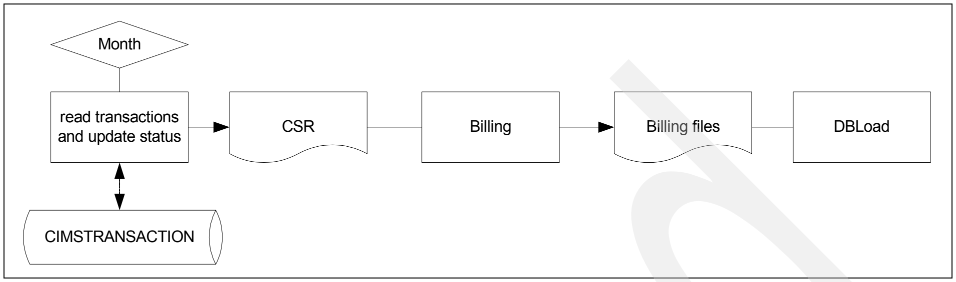


Figure 8-9 The transaction function overview

Feeding the table can be done via the ISC menu selections **Usage and Accounting Manager** → **System Maintenance** → **Transactions**, or directly using SQL from other database sources.

On the Transactions panel returned (Figure 8-10) select **Credit** and click the **New** button. The panel for adding transactions is displayed (Figure 8-11 on page 267).

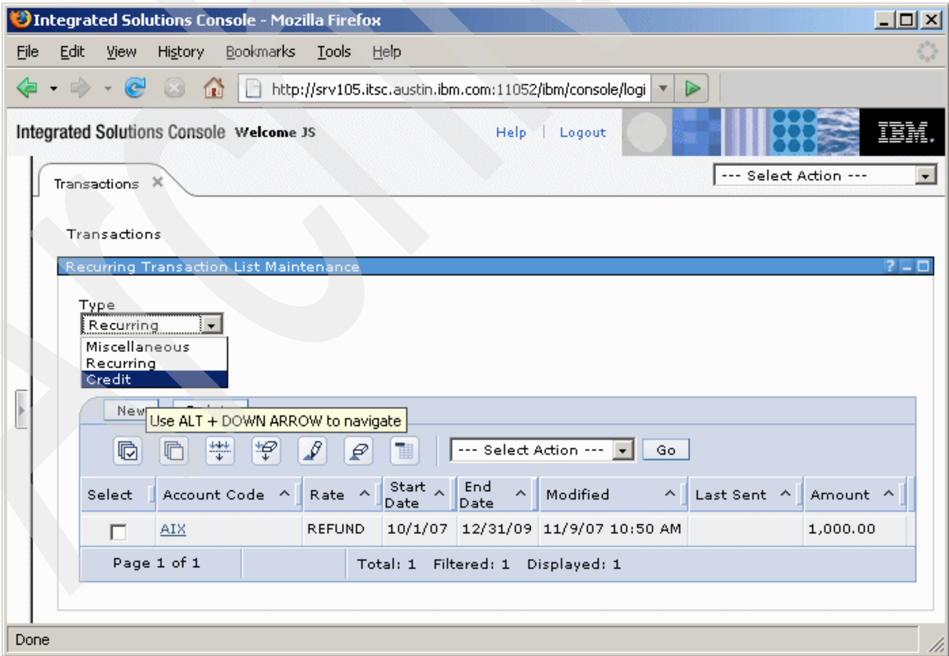


Figure 8-10 The ISC Transactions menu

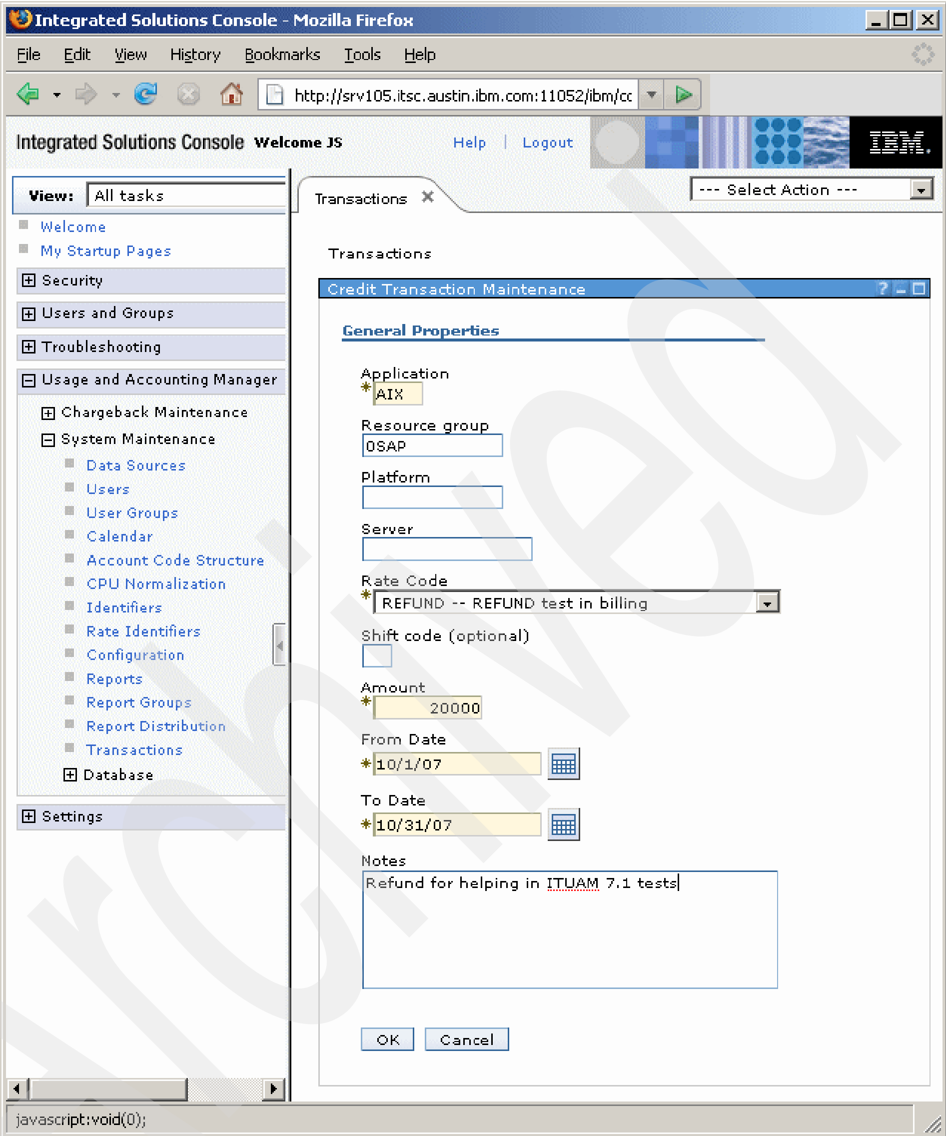


Figure 8-11 Adding a Transaction using ISC

Notice that only negative rate codes are displayed in the credit panel.

Figure 8-12 on page 268 shows the table structure used as input for the transaction job file.

```
COLNAME                   TYPE       LENGTH
------------------------- ---------- -----------
TRANSACTIONUID            GRAPHIC             32
ACCOUNTCODE               GRAPHIC            127
TRANSACTIONTYPE           GRAPHIC              1
SHIFTCODE                 GRAPHIC              1
RATECODE                  GRAPHIC              8
RESOURCEAMOUNT            DECIMAL             18
FREQUENCY1                INTEGER              4
FREQUENCY2                INTEGER              4
FROMDATE                  TIMESTAMP           10
TODATE                    TIMESTAMP           10
DATETIMESENT              TIMESTAMP           10
DATETIMEMODIFIED          TIMESTAMP           10
DATETIMEENTERED           TIMESTAMP           10
DATETIMESTARTPROCESSING   TIMESTAMP           10
DATETIMESTOPPROCESSING    TIMESTAMP           10
USERID                    VARGRAPHIC         255
DATETIMEDELETED           TIMESTAMP           10
NOTE                      VARGRAPHIC         255
```

Figure 8-12 The table structure of CIMSTRANSACTION

The columns have the following meanings:

TRANSACTIONUID Unique number to identify the transaction.

ACCOUNTCODE Must match the account code structure set up in IBM Tivoli Usage and Accounting Manager.

TRANSACTIONTYPE Defined by the first character of this types:

- Recurring Monthly service fees (such as leased line charge)
- Credit Refund on overcharges
- Miscellaneous All kinds of one-time charges (such as initial setup)

SHIFTCODE Shift code, if applicable.

RATECODE Depending on the rate definition, negative rates should be used with TRANSACTIONTYPE “C” or “R” and positive rates with either “M” or “R”.

RESOURCEAMOUNT The amount of resources to be charged.

FREQUENCY1 & 2 Not used with ITUAM 7.1; all transactions are monthly.

FROMDATE/TODATE Date range for TRANSACTIONTYPE “C” and “M” only.

DATETIMESTARTPROCESSING Start date for TRANSACTIONTYPE “R”.

DATETIMESTOPPROCESSING Stop date for TRANSACTIONTYPE “R”.

NOTE A short description of the transaction.

All other “DATETIME” columns are used for timestamps on manipulation and processing of the transactions.

The extract from the transaction job file in Example 8-1 shows the updated dataSourceName and the specific LogDate parameter for this job. We created it by copying from the sample job files.

```
cd /opt/ibm/tuam/jobfiles
cp ../samples/jobfiles/SampleTransaction.xml LoadTransaction.xml
```

Example 8-1 The LoadTransaction.xml job file updates required

```
<Input name="CollectorInput" active="true">
  <Collector name="TRANSACTION">
    <Connection dataSourceName="ITUAMDB"/>
  </Collector>

<!-- This is the LogDate parameter for the collector. Valid values are:
PREMON, CURMON, or a date string in YYYYPP format. -->
  <Parameters>
    <Parameter name="LogDate" value="PREMON" DataType="String"/>
  </Parameters>
</Input>
```

Running the job from the command line will load the data into the database and update the transaction timestamps in the database table.

```
/opt/ibm/tuam/bin/startJobRunner.sh LoadTransaction.xml
```

Check the invoice report (Figure 8-13 on page 270). The refund has been subtracted from the billing.

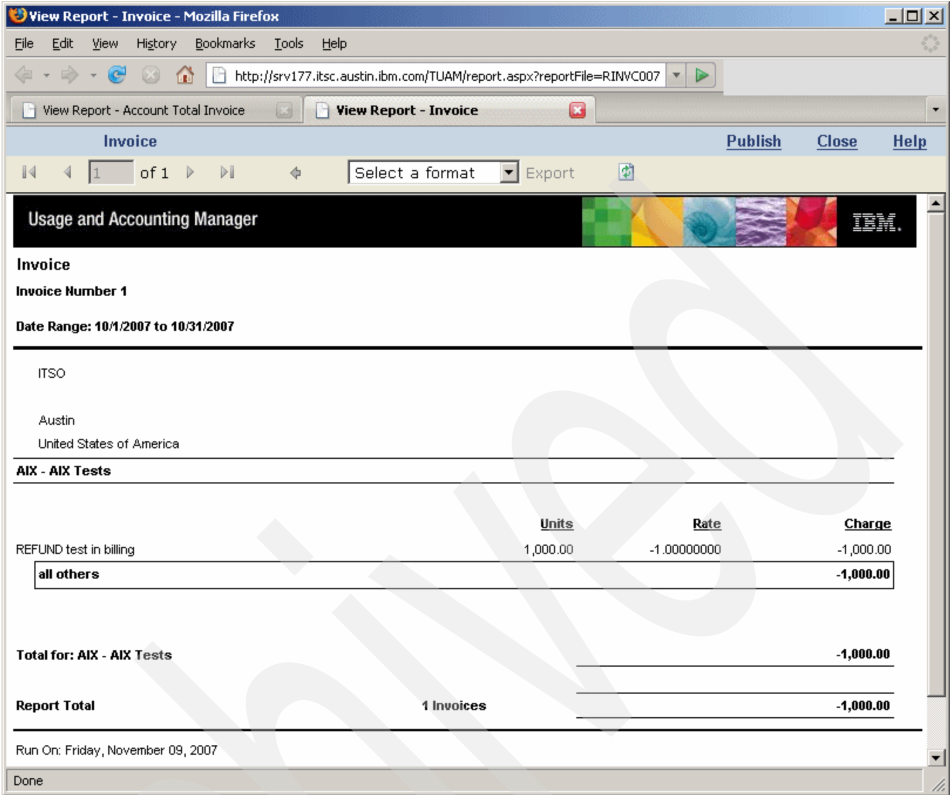


Figure 8-13 An Invoice with refund from Transaction function

8.3 Database size collection

The size of the Tivoli Usage and Accounting Manager database is dependent on many different parameters. Estimating the storage requirements for the database is therefore a difficult task. Still, at some time we need to do some kind of estimate. That could be a pure guess, or one that is based on previous experiences. For our environment we have developed a space tracking tool that will help us make space estimates based on the actual table sizes over time.

In the appendix, “Sample script tuamdbsize.sh” on page 356 provides a script that will create two types of reports:

- A table size report, which shows the number of rows and the size in KB used for each of the Tivoli Usage and Accounting Manager tables.

- A load tracking report, which shows the number of records that have been loaded, but not deleted, for each of the Tivoli Usage and Accounting Manager Source feeds and file types.

In the report in Example 8-2 we see that three of the tables account for most of the space used.

Example 8-2 Table size report

```
Thu Nov  8 15:13:41 CST 2007
T a b l e   n a m e                         R o w s   S i z e KB  Rows/MB
CIMSDETAILIDENT                             3346635      90360    37037
CIMSDETAIL                                   823215     183577     4484
CIMSSUMMARY                                   11217       3276     3424
CIMSLOADTRACKING                               1015         72    14285
CIMSRATEIDENTIFIERS                             860         11    83333
CIMSRATETORATEGROUP                             493          6    83333
CIMSCALENDAR                                    492         19    26315
CIMSRATE                                        483        100     4878
CIMSSUMMARYTODETAIL                             355         11    33333
CIMSREPORTTOREPORTGROUP                          88          2    76923
CIMSREPORT                                       64          5    12820
CIMSCONFIGOPTIONS                                52          5    12987
CIMSIDENT                                        50          2    31250
CIMSRATEGROUP                                    46          3    20000
CIMSREPORTGROUP                                  16          1    27777
CIMSRATESHIFT                                    16          2    15151
CIMSCONFIGACCOUNTLEVEL                           13          1    16949
CIMSREPORTDISTRIBUTIONTYPE                       11          1    26315
CIMSCLIENT                                       11          4     3344
CIMSCPUNORMALIZATION                              9          1    13513
CIMSREPORTDISTRIBUTIONPARM                        7          1    27027
CIMSTRANSACTION                                   5          2     3205
CIMSUSERTOUSERGROUP                               4          1    62500
CIMSUSER                                          4          1    18518
CIMSUSERGROUPCONFIGOPTIONS                        3          1    14084
CIMSUSERGROUPACCOUNTSTRUCTURE                     3          1    22727
CIMSREPORTSTART                                   3          1   125000
CIMSUSERGROUP                                     1          1    31250
CIMSUSERGROUPACCOUNTCODE                          1          1     7407
CIMSREPORTDISTRIBUTION                            1          1    31250
CIMSREPORTDISTRIBUTIONCYCLE                       1          1    83333
CIMSCONFIG                                        1          1     3584
CIMSUSERGROUPREPORT                               0          0
CIMSUSERFAVORITES                                 0          0
CIMSUSERCONFIGOPTIONS                             0          0
```

```
CIMSSUMMARYDAILY                              0             0
CIMSRESOURCEUTILIZATION                       0             0
CIMSREPORTCUSTOMFIELDS                        0             0
CIMSHEADLINE                                  0             0
CIMSCLIENTCONTACT                             0             0
CIMSCLIENTCONTACTNUMBER                       0             0
CIMSCLIENTBUDGET                              0             0
T o t a l                               4185175        277472    15083
```

The size of these three tables also relates directly to information in messages from the Database Load step. In Example 8-3 we see the number of records (rows) added to the three tables as a result of processing the 65 CSR records.

Example 8-3 Database Load messages extract

```
06:37:14.852: INFORMATION  Summary Load: Load Started
06:37:14.938: INFORMATION  Loaded Records: 97
06:37:14.941: INFORMATION  Summary Load: Load Completed Successfully
06:37:14.943: INFORMATION  Detail Load: Load Started
06:37:15.524: INFORMATION  Loaded Records: 65    Resources 257
06:37:15.524: INFORMATION  Detail Load: Load Completed Successfully
06:37:15.528: INFORMATION  Ident Load: Started
06:37:15.596: INFORMATION  Loaded Records: 352
06:37:15.605: INFORMATION  Ident Load: Load Completed Successfully
06:37:15.610: INFORMATION  Number of Detail Records Loaded: 257
06:37:15.610: INFORMATION  Number of Ident Records Loaded: 352
06:37:15.610: INFORMATION  Number of Summary Records Loaded: 97
06:37:15.610: INFORMATION  DBLoad Completed Successfully
```

From this we can *estimate* the increased size (MB) after the Database Load by dividing the loaded records by the “Rows/MB” from Example 8-2 on page 271:

- CIMSDETAIL: 257 / 4484 = 0.057
- CIMSDETAILIDENT: 352 / 37037 = 0.010
- CIMSSUMMARY: 97 / 3434 = 0.028

Example 8-4 shows the second type of report the load tracking report.

Example 8-4 Load track report

```
Thu Nov  8 15:13:41 CST 2007
S o u r c e f e e d    F i l e t y p e   R e c o r d s
AATRID1                Detail                   585511
AATRID1                Ident                   2445431
AATRID1                Summary                    1523
```

AATRID4	Detail	4016
AATRID4	Ident	2700
AATRID4	Summary	1872
AATRID8	Detail	203275
AATRID8	Ident	847245
AATRID8	Summary	671
TDSzADRL	Detail	2491
TDSzADRL	Ident	5376
TDSzADRL	Summary	962
TDSzBAT	Detail	1354
TDSzBAT	Ident	2812
TDSzBAT	Summary	499
TDSzJOBL	Detail	1402
TDSzJOBL	Ident	3162
TDSzJOBL	Summary	507
TDSzSESL	Detail	881
TDSzSESL	Ident	1799
TDSzSESL	Summary	233
TDSzSTC	Detail	490
TDSzSTC	Ident	938
TDSzSTC	Summary	177
TDSzTSO	Detail	494
TDSzTSO	Ident	957
TDSzTSO	Summary	76
TDSzUSER	Detail	8387
TDSzUSER	Ident	11408
TDSzUSER	Summary	2745
TRANSR00	Detail	6
TRANSR00	Ident	36
TRANSR00	Summary	6
UNIXFSYS	Detail	10748
UNIXFSYS	Ident	17481
UNIXFSYS	Summary	501
UNIXSPCK	Detail	2653
UNIXSPCK	Ident	4770
UNIXSPCK	Summary	138
VMWARE	Detail	1507
VMWARE	Ident	2520
VMWARE	Summary	1307
T o t a l		4181067

The estimate done so far is based on an average table row size and is a result of collecting from many different sources. To make a more accurate estimate we need to do the following:

1. Run the ituamdbsize.sh script and take note of the size of the three tables highlighted in Example 8-2 on page 271.
2. Run a Job File. From the Scan step in the Log file you will find the number of CSR records created.
3. Run the ituamdbsize.sh script again and calculate the space increase.

The results we obtained by running these scripts for various job files are shown in Table 8-1.

Table 8-1 Database size increase statistics

Job File	Run #	CSR Records	Database size increase (KB)	Database size per 1000 CSR records (MB)
TDSzLoad1.xml	1	73	108	1.48
	2	756	904	1.20
	3	19	22	1.16
LoadTDSzUser2.xml	1	63	105	1.67
	2	804	1124	1.40
	3	288	402	1.40
LoadAIXAA05.xml	1	72	180	2.50
	2	103	190	1.84
	3	168	198	1.18
	4	198	307	1.55
LoadVIOSAP.xml	1	12488	12070	0.97
	2	12491	12099	0.97
	3	14	13	0.93

In the appendix we also provide a statistical script (“Sample script tuamdbstat.sh” on page 359) to calculate the daily growth based on a daily scheduled tuamdbsize script output. Figure 8-14 on page 275 shows sample output of the script.

```
[db2inst1@srv105 ITUAMDB]$ cat 20071116-streport.txt
T i m e s t a m p          R o w s   S i z e KB     Growth KB       Rows/MB
20071101165701             1583940       98589
20071102152500             1984634      121836         23247          16289
20071103080707             2155805      132140         10304          16314
20071104080707             2326059      142341         10201          16341
20071105103950             2697273      164638         22297          16383
20071106080818             3170653      194958         30320          16263
20071107132557             3943600      261350         66392          15089
20071108151341             4185175      277472         16122          15083
20071109080710             4355770      288395         10923          15103
20071110080927             4525195      299169         10774          15125
20071111080710             4693776      309832         10663          15149
20071112080710             4862219      320479         10647          15171
20071113080713             5290676      348351         27872          15187
20071114080758             5459856      359083         10732          15204
20071115080819             5717477      375669         16586          15219
20071116103341             5773776      378298          2629          15262
```
Figure 8-14 Sample output of the tuamdbstat.sh script

9

Troubleshooting tips

This chapter provides some tips on problem solving while using Tivoli Usage and Accounting Manager. The problems addressed are:

9.1 General logging and tracing options

The logging and tracing settings for Tivoli Usage and Accounting Manager are stored in the logging.properties configuration file, which is located in /opt/ibm/tuam/config.

The logging.properties file can be accessed from the Configuration page shown in Figure 9-1. To access this file, go to the Integrated Solution Console (ISC) Web interface and select **Usage and Accounting Manager** → **System Maintenance** → **Configuration** → **Logging**.

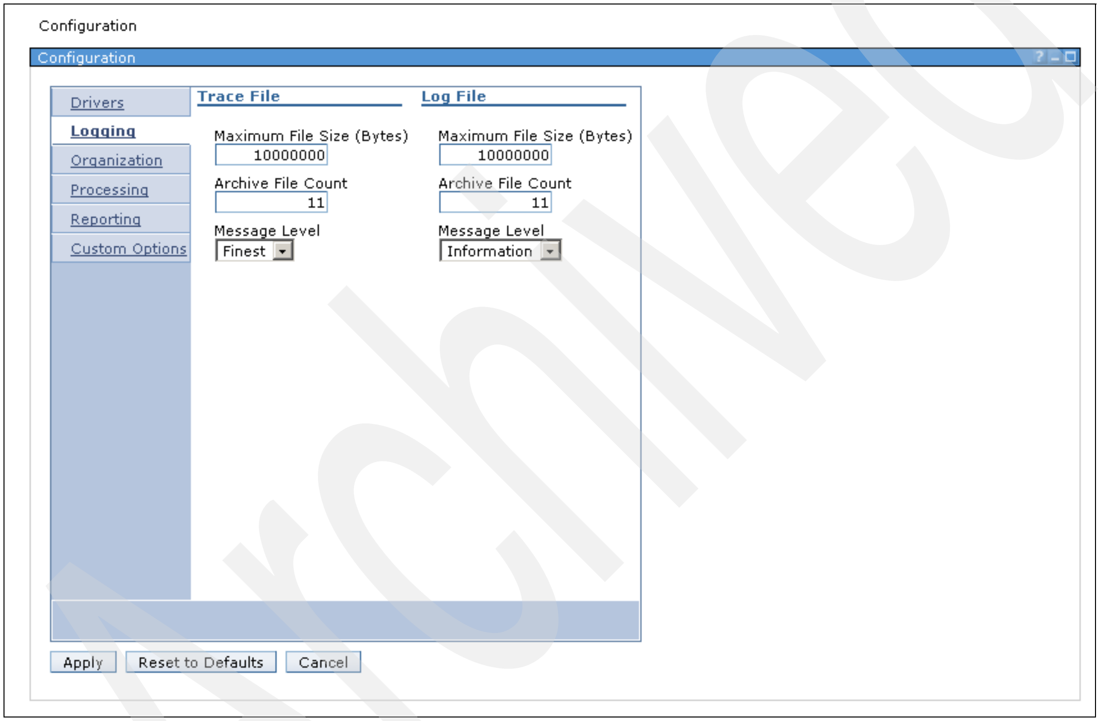


Figure 9-1 Logging options

You can set the file size for tracing and logging files, number of generations, and logging levels. These trace files are written to the /opt/ibm/tuam/logs/server directory.

Our sample logging.properties file is shown in Example 9-1.

Example 9-1 Sample logging.properties

```
#Oct 31, 2007 12:16:08 PM
handlers=com.ibm.tivoli.ituam.logger.MessageFileHandler,com.ibm.tivoli.
ituam.logger.TraceFileHandler
.level=FINEST
com.ibm.tivoli.tuam.logger.MessageFileHandler.append=true
com.ibm.tivoli.tuam.logger.MessageFileHandler.count=11
com.ibm.tivoli.tuam.logger.MessageFileHandler.formatter=java.util.loggi
ng.SimpleFormatter
com.ibm.tivoli.tuam.logger.MessageFileHandler.level=INFO
com.ibm.tivoli.tuam.logger.MessageFileHandler.limit=10000000
com.ibm.tivoli.tuam.logger.MessageFileHandler.pattern=C:/ibm/tuam/logs/
server/message%g.log
com.ibm.tivoli.tuam.logger.TraceFileHandler.append=true
com.ibm.tivoli.tuam.logger.TraceFileHandler.count=11
com.ibm.tivoli.tuam.logger.TraceFileHandler.formatter=java.util.logging
.SimpleFormatter
com.ibm.tivoli.tuam.logger.TraceFileHandler.level=FINEST
com.ibm.tivoli.tuam.logger.TraceFileHandler.limit=10000000
com.ibm.tivoli.tuam.logger.TraceFileHandler.pattern=C:/ibm/tuam/logs/se
rver/trace%g.log
```

As indicated in Example 9-1, the settings are for the message file and trace file. The settings include:

append Whether to append to the log files after a restart

count Number of generations of the log file

formatter Log file formatter class

level Level of logging to be recorded in this type of log

limit File size limit, before a new generation is created

pattern File name of the log file, the default is using Trace%g.log or Message%g.log (%g indicates the generation number)

The trace and log files are written from the ISC and job processes. Every time a process accesses the trace or message file, a lock file (.lck) is created. If another process wants to write to a log file, it will create an additional trace file with a numbered suffix.

The trace and message log file names are in the format of <type><n>.log.<m>; where:

type — Message or trace
n — Archived log file serial number; the current log has the serial of 0.
m — Number entries for different processes that writes log files

The reporting application uses a different log file called trace_net0.log. This is generated from the application under the Microsoft Internet Information Server.

9.2 Installation and configuration problems

The installation process has a different logging default than the program itself. It is typically the Tivoli common logging directory; which in Windows is \Program Files\ibm\tivoli\common\AUC\logs\install; however, in UNIX it is /opt/ibm/tivoli/common/AUC/logs/install.

The log file for the Enterprise Edition and Enterprise Collector Pack is called TUAMInstall.log. In Windows, there is an additional DB2RTCInstall.log file for the DB2 UDB V9.1 runtime client. The Windows Process Collector creates an additional log file called WPCInstall.log.

The stages of installation for the Enterprise Edition are performed mostly from the setup/console directory:

1. The files are transferred into the installation directory.
2. The wizard installs an embedded WebSphere Application Server and the Integrated Solution Console. This is done by simply unzipping the EmbeddedExpress and ISCAE71 zip files.
3. The wizard deploys the Tivoli Usage and Accounting Manager application using the deployTUAM.bat command to run deployTUAMConsole.py that installs the aucConsole.war.
4. In Windows only, the wizard invokes **`db2rtc.bat`** to install the DB2 runtime client.
5. Post installation is performed using the tuamPostInstall.bat. In Windows, the installation wizard installs the report application using iisconfig.vbs, and the FinancialModeler using financialModelerConfig.bat.

9.3 Integrated Solution Console debugging

The Integrated Solution Console is based on a WebSphere Application Server. Apart from the standard Tivoli Usage and Accounting Manager logging files, some information can be retrieved also from the WebSphere logs.

WebSphere logs are the standard output and errors of the WebSphere's JVM. These are in $TUAM_home/ewas/profiles/AppSrv01/logs/server1 with the file names of SystemOut.log and SystemErr.log respectively.

Doing configuration tasks, you might get a message like that shown in Figure 9-2. If you do, check for the Tivoli Usage and Accounting Manager server logs first. Then you might need to check your database logs. In certain cases it might help to watch for the WebSphere logs, to get some information on connectivity.

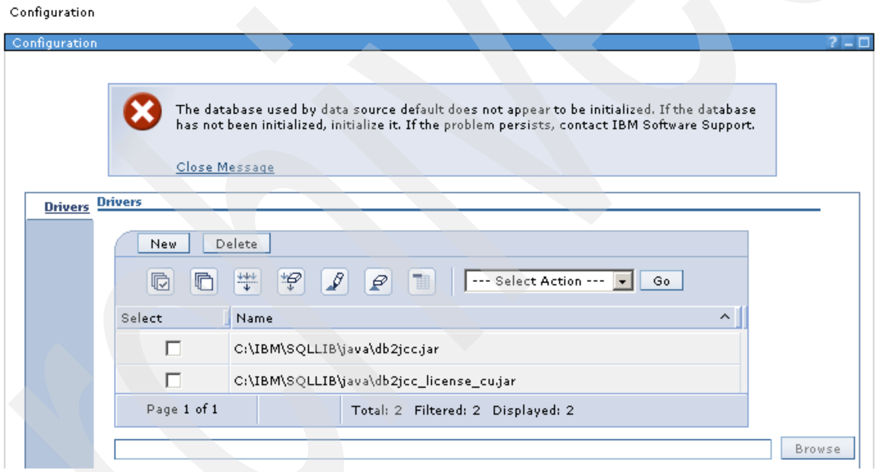


Figure 9-2 Error message on database task

For some messages (see Figure 9-3 on page 282) you may not need to watch for details in the log.

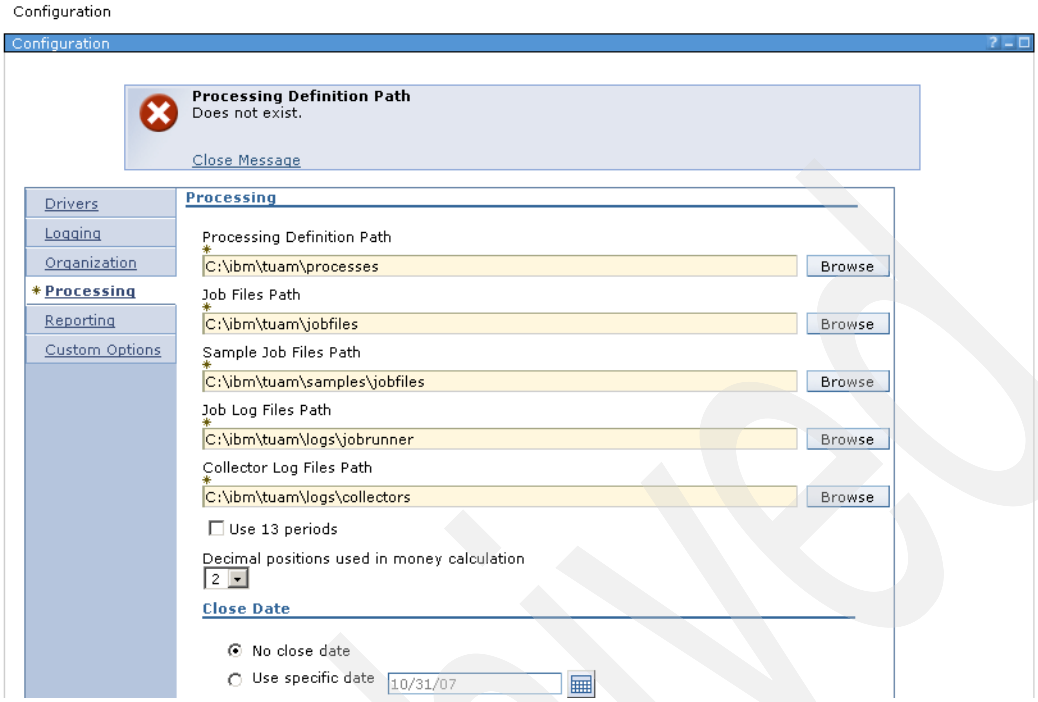


Figure 9-3 Error message on configuration task

9.4 Job runner debugging

For the Tivoli Usage and Accounting Manager processing engine the trace option can be set in addition within the XML job files on step and stage level.

- Acct step parameter trace=”true”
- Bill step parameter trace=”true”
- Integrator step will set it on stage level: <Stage name="*function*" trace="true">
- For Integrator collector section use <parameter name=”trace” value=”on” />

> **Note:** The trace options are not consistent, so you can try using upper or lower case and **ON** instead of **true** in some cases.

Two types of output are produced when a job is running:

- A Job Runner log file, which is located in the /opt/ibm/tuam/logs/jobrunner directory in a directory named according to the “Job Id” parameter value in the job file. The XML version is for the ISC to display the log file and the text version can be used for searching on the command line level or viewing with an editor.

- The Trace and message files, located in the /opt/ibm/tuam/logs/server directory, are active for the entire life of the application server running under embedded WebSphere Application Server.

Running a job from the ISC, failures will display an error message as shown in Figure 9-4.

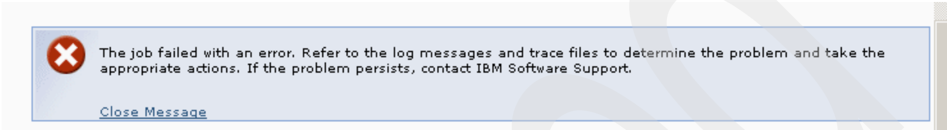


Figure 9-4 Error message due to job failure

For more details we can search the log files, like the one shown in Figure 9-5.

```
[root@srv105 /]# cd /opt/ibm/tuam/logs/jobrunner/AIXAA_aggregated
[root@srv105 AIXAA_aggregated]# ls -tr *txt | tail -3 | while read file ; do grep -E
.*AUCJR003[1-2].* $file; done
11/5/07 13:32:11.197: INFORMATION    AUCJR0032I The job AIXAA_aggregated completed at
Nov 5, 2007 1:32:11 PM with 1 warning, 0 errors.
11/5/07 13:52:47.560: INFORMATION     AUCJR0031I The AIXAA_aggregated process
completed successfully at the following time: Nov 5, 2007 1:52:47 PM.
11/5/07 13:53:44.934: INFORMATION    AUCJR0032I The job AIXAA_aggregated completed at
Nov 5, 2007 1:53:44 PM with 0 warnings, 1 error.
[root@srv105 AIXAA_aggregated]# ls -tr *txt | tail -1 | while read file ; do echo
$file ; grep -i warn $file | wc -l; grep -i error $file | wc -l ; done
20071105_135342.txt
4 # shows the # of warnings
8 # shows the # of errors
```

Figure 9-5 Searching the logs on the command line level

For detailed analysis of the last log, you can issue the command:

```
ls -tr | tail -1 | while read file; do more $file; done
```

9.5 Quick finder for trace and log information

Table 9-1 on page 284 is a summary of where to change settings and search for files.

Table 9-1 Overview for trace and log files

Path or filename	Function
/opt/ibm/tuam/logs/jobrunner/<JobId> Job runner log files separated per JobID	
<timestamp>.txt	Job log output
<timestamp>.xml	Job log for use with the ISC
/opt/ibm/tuam/logs/server/ Tivoli Usage and Accounting Manager trace and log files	
message0.log message<g>.log<#>	Messages from tuam processing, where <g> = generation and <#> = instance
trace0.log trace<g>.log<#>	Trace details for tuam processing, where <g> = generation and <#> = instance
*.lck	Lock files for trace and log coordination
trace_net0.log	Trace for the reporting server on Windows only
/opt/ibm/tuam/ewas/profiles/AppSrv01SystemOut.log/logs/server1 WebSphere and Integrated Solutions Console (ISC) files	
SystemOut.log	WebSphere messages
SystemErr.log	WebSphere error log
/opt/ibm/tuam/config Tivoli Usage and Accounting Manager config files	
logging.properties	Settings for trace and message files
jdk_logging.properties	Not used with version 7.1
/opt/ibm/tivoli/common/AUC/logs/install Installation and uninstallation log files	

A

Sample listings and programs

This appendix contains the following samples:

Sample job for remote deployment to AIX

```
<?xml version=”1.0” encoding=”utf-8”?>
<!--
 ************************************************************ {COPYRIGHT-TOP}
 * Licensed Materials - Property of IBM
 * IBM Tivoli Usage and Accounting Manager
 * 5724-O33, 5765-UAV, 5765-UA7, 44E7863
 * (c) Copyright IBM Corp. 2004, 2007
 *
 * The source code for this program is not published or otherwise
 * divested of its trade secrets, irrespective of what has been
 * deposited with the U.S. Copyright Office.
 ************************************************************ {COPYRIGHT-END}
-->
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
  <Job    id=”DeployAIX”
          description=”Deploy the ITUAM UNIX/Linux Collector Agent”
          active=”true”
          joblogShowStepParameters=”true”
          joblogShowStepOutput=”true”
          processPriorityClass=”Low”
          joblogWriteToTextFile=”true”
          joblogWriteToXMLFile=”true”
          smtpSendJobLog=”false”
          smtpServer=”mail.ITUAMCustomerCompany.com”
          smtpFrom=”ITUAM@ITUAMCustomerCompany.com”
          smtpTo=”John.ITUAMUser@ITUAMCustomerCompany.com”
          stopOnProcessFailure=”false”>
    <Process    id=”DeployUnixLinuxCollector”
                description=”Deployment of ITUAM UNIX/Linux Data Collector”
                joblogShowStepOutput=”true”
                joblogShowStepParameters=”true”
                active=”true”>
      <Steps stopOnStepFailure=”true”>
        <Step   id=”ITUAM UNIX/Linux Data Collector Deployment”
                description=”ITUAM UNIX/Linux Data Collector Deployment”
                type=”ConvertToCSR”
                programName=”rpd”
                programType=”java”
                active=”true”>
          <Parameters>
            <Parameter Action           = “install”/>
            <!-- SUPPLY hostname OF TARGET PLATFORM/-->
```

```
            <Parameter Host             = “lparxx”/>
            <!-- userid must be set to root/-->
            <Parameter UserId           = “root”/>
            <!-- SUPPLY root PASSWORD ON TARGET PLATFORM/-->
            <Parameter Password         = “passw0rd”/>
            <Parameter KeyFilename     = “/root/.ssh/id_rsa”/>
            <!-- DEFINE Manifest TO MANIFEST XML FOR TARGET PLATFORM/-->
            <Parameter Manifest         = “DeploymentManifest_aix5.xml”/>
            <!-- DEFINE INSTALLATION PARAMETERS,
                 path:   must be defined to the directory path where UNIX/Linux
Collector
                        will be installed on target platform
             /-->
            <Parameter RPDParameters    =
“path=/opt/ibm/tuam/collectors/Unix;user=root;”/>
            <Parameter Verbose          = “true”/>
            <Parameter SourcePath       = “%HomePath%/collectors/unixlinux”/>
          </Parameters>
        </Step>
      </Steps>
    </Process>
  </Job>
</Jobs>
```

Sample script to transfer UNIX data

```
#!/bin/sh
#
###################################################################
# Created by Roy 20071016 based on CS_send
# 1. Updated the SCP commands to perform “pull” from the AIX server
###################################################################
# Licensed Materials - Property of IBM
#
# “Restricted Materials of IBM”
#
# IBM Tivoli Usage and Accounting Manager
# 5724-O33, 5765-UAV, 5765-UA7, 44E7863
# (c) Copyright IBM Corp. 1996,2004, 2007
#
# US Governmant Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with
# IBM Corp.
```

```
#################################################################
#
#     Module: CS_pull
#
#  The Common Source Send Command Procedure sends Nightly Input Source Files
#   containing CSR Records to the ITUAM Server platform and places the files in
#   the appropriate PROCESSES folder for the Input Source on the ITUAM Server
#   Platform.
#
#  Note: Nightly Input Source Files contain the date on which they were
#        generated in their filename. For example, CS_sum_20030702.csv.
#        The data contained in the file is usage information for the previous
#        day. For reprocessing compliance with ITUAM Server, this script will
#        rename the files with their usage date in the filename on the ITUAM
#        Server platform.
#
# For example:
#
#                    Filename on the                       Filename in the
#                Unix Consolidation Server          ITUAM Server PROCESSES folder
#               -----------------------------       -----------------------------
#
#CS_sum_20030702.csv        ->   UnixOS\20030701.txt
#CS_sum_fs_20030702.csv     ->   UnixFS\20030701.txt
#CS_sum_ora_20030702.csv    ->   UnixORA\20030701.txt
#CS_sum_orasto_20030702.csv ->   UnixORAstorage\20030701.txt
#CS_sum_db2_20030702.csv    ->   UnixDB2\20030701.txt
#CS_sum_db2sto_20030702.csv ->   UnixDB2storage\20030701.txt
#
#  This script should be run some time after CS_nightly_consolidation has
#   completed.
#
#  CS_send will take the values of the following variables defined in
#   the ITUAM/UNIX Configuration File, $ITUAM_HOME/data/A_config.par, to determine
the
#   Input Source Files to be transferred to the ITUAM Server platform.
#
#GEN_UNIXFS     : Transfer Unix File System Input Source File
#GEN_ORA        : Transfer ORACLE Input Source File
#GEN_DB2        : Transfer DB2 Input Source File
#GEN_ORA_STORAGE: Transfer ORACLE Storage Input Source File
#GEN_DB2_STORAGE: Transfer DB2 Storage Input Source File
#
#  CS_send will use the values of CS_METHOD, CS_USER, CS_KEY, CS_UPATH, and
#   CS_PLATFORM from the ITUAM/UNIX Configuration File,
```

```
#   ($ITUAM_HOME/data/A_config.par), to access the ITUAM Server Platform unless
#   overridden in this script.
#
#       CS_METHOD   : Transfer method, FTP, SFTP, or SCP.
#CS_USER     : Username of account on the ITUAM Server Platform
#       CS_KEY      : Password for user account on ITUAM Server Platform
#       CS_UPATH    : Home directory of user running this script.
#       CS_PLATFORM : Name of the ITUAM Server Platform.
#
#CS_PROC_PATH : Path to the PROCESSES folder on the ITUAM Server
#                     platform. If the PROCESSES folder is the Default FTP
#                     home, leave this variable blank.
#
#  CS_METHOD defines the file transfer protocol that will be used.
#
#   All values of CS_METHOD assume that CS_PLATFORM is defined to the
#     ITUAM Server platform. Also CS_PROC_PATH should be defined if the
#     default login directory for CS_METHOD is not the ITUAM Server
#     PROCESSES folder.
#
#   NOTE: If CS_METHOD is set to SFTP and CS_PROC_PATH is not blank, be sure
#     to use forward slashes when defining the path to the ITUAM Server Processes
#     folder. For example...
#
#CS_PROC_PATH=C:/ITUAM/PROCESSES
#
#     If CS_METHOD is set to SCP, CS_PROC_PATH must be initialized, be sure
#     to use backward slashes when defining the path to the ITUAM Server Processes
#     folder. For example...
#
#CS_PROC_PATH=C:\ITUAM\PROCESSES
#
#   If CS_METHOD is set to FTP, you will need to define CS_USER, CS_KEY
#     CS_UPATH and possibly CS_PROC_PATH.
#
#   If CS_METHOD is set to SFTP, you will need to define CS_USER and ensure
#     that a Secure Shell Public Key has been generated for ITUAM_USER to
#     connect to CS_PLATFORM as CS_USER without using a passphrase. This
#     script is to be run as root.
#
#   If CS_METHOD is set to SCP, you will need to define CS_USER.
#     CS_PROC_PATH must be defined as the path to the ITUAM Server PROCESSES
#     folder. Ensure that a Secure Shell Public Key has been generated for
#     ITUAM_USER to connect to CS_PLATFORM as CS_USER without using a
#     passphrase. This script is to be run as root.
```

```
#     NOTE: CS_METHOD=SCP is only valid if CS_PLATFORM is a Windows platform.
#
#   If CS_METHOD is set to MV, the input source files are to be moved to
#     process directories pointed to by CS_PROC_PATH. Using MV, indicates
#     you are loading ITUAM Server Database from this Unix platform.
#
#   The following variables in A_Config.par define the PROCESSES sub-directory
#   names where input source files will be delivered.
#
#   variableDefault value
#   ---------------------
#CS_UNIXOS_PROCDIRUnixOS
#CS_UNIXFS_PROCDIRUnixFS
#CS_ORA_PROCDIRUnixORA
#CS_ORA_STORAGE_PROCDIRUnixORAstorage
#CS_DB2_PROCDIRUnixDB2
#CS_DB2_STORAGE_PROCDIRUnixDB2storage
#
# Usage:
#> CS_send [date] [input_type]
#
#If called with no arguments, all input source files for the current
#day are sent to the ITUAM Server platform.
#
#date - in the format YYYYMMDD
#
#input_type_list - Any combination of
#SUM UNIXFS ORA ORASTO DB2 DB2STO
#
# Examples:
#
#> CS_pull Send all input source files with todays
#                                       date to the ITUAM Server Platform. Files
#                                       will be renamed to the previous day and
#                                       placed in the appropriate PROCESSES folder.
#
#> CS_pull 20030720Send all input source files with 20030720
#                                       in their filename to the ITUAM Server
#                                       Platform. Files will be named 20030719.txt
#                                       and placed in the appropriate PROCESSES
#                                       folder.
#
#> CS_pull 20030720 ORA ORASTO   Send ORACLE and ORACLE Storage input
#source files for 20030720 to the ITUAM
#                                       Server Platform. Files will be named
```

```
#                                       20030719.txt and placed in the
#                                       appropriate PROCESSES folder.
#
#  CS_pull Modification Log
#
#       GCB V01.01 02-Jul-2003
#
#               Initial creation of this routine.
#
#GCB V01.02 06-Aug-2003
#
#Added CS_METHOD variable.
#
#GCB V01.03 14-Nov-2003
#
#Added support for CS_PROC_PATH variable, which is read from the
#ITUAM/UNIX Configuration file. This variable should be set to the path
#from the FTP DEFAULT directory on the ITUAM Server platform to the
#ITUAM Server Processes folder.
#
#GCB V01.04 26-Nov-2003
#
#Support for CS_METHOD to be set to SFTP and SCP. See notes above
#for decription of the environment variables that must to be
#initialized when using these values of CS_METHOD.
#
#GCB V01.05 24-Feb-2005
#
#Support for CS_METHOD set to MV.
#
#GCB V01.06 02-Jan-2006
#
#Added support for reading PROCDIRs from A_config.par.
#
#GCB V01.07 27-Feb-2006
#
#Support new filenames in ITUAM release.
#
#GCB V01.08 21-Jul-2006
#
#Corrected syntax error in sftp call.
#
#GCB V02.01 25-Jan-2007
#
#Added ICU messaging.
```

```
#
#

define_destination()
{
   case $file_item in
    CS_sum_fs_*)
       if test “$CS_UNIXFS_PROCDIR” != ““
       then
           DESTIN=”$CS_UNIXFS_PROCDIR”
       else
           DESTIN=”UnixFS”
       fi
     ;;
    CS_sum_orasto_*)
       if test “$CS_ORA_STORAGE_PROCDIR” != ““
       then
           DESTIN=”$CS_ORA_STORAGE_PROCDIR”
       else
           DESTIN=”UnixORAstorage”
       fi
     ;;
    CS_sum_ora_*)
       if test “$CS_ORA_PROCDIR” != ““
       then
           DESTIN=”$CS_ORA_PROCDIR”
       else
           DESTIN=”UnixORA”
       fi
     ;;
    CS_sum_db2sto_*)
       if test “$CS_DB2_STORAGE_PROCDIR” != ““
       then
           DESTIN=”$CS_DB2_STORAGE_PROCDIR”
       else
           DESTIN=”UnixDB2storage”
       fi
     ;;
    CS_sum_db2_*)
       if test “$CS_DB2_PROCDIR” != ““
       then
           DESTIN=”$CS_DB2_PROCDIR”
       else
           DESTIN=”UnixDB2”
       fi
```

```
     ;;
    CS_sum_*)
       if test "$CS_UNIXOS_PROCDIR" != ""
       then
           DESTIN="$CS_UNIXOS_PROCDIR"
       else
           DESTIN="UnixOS"
       fi
     ;;
    *)
     echo "ERROR: ***** CS_pull Invalid Argument ${arg} "
     ;;
  esac
  export DESTIN
}

#
# CS_pull initialization
#

# test for presence of ITUAM/UNIX configuration file (/etc/ituam_uc.conf)
# if this file is missing, a step was missed during installation;
# this matter must be corrected before proceeding

if test ! -f /etc/ituam_uc.conf
then
    echo "ERROR **** CS_pull: ITUAM/UNIX Configuration file (/etc/ituam_uc.conf) not
found"
    exit 1
fi

#
# test for presence of the ITUAM/UNIX environment configuration file
# ($ITUAM_DATA/A_config.par); this file enables various environment
# variables to be defined; if this file is missing, a step was missed
# during installation; this file should be created

: ${ITUAM_DATA:=`awk -F= '/^ITUAM_DATA/ {print $2}' /etc/ituam_uc.conf`}

if test -z "${ITUAM_DATA}"
then
    echo "ERROR **** CS_pull: ITUAM_DATA environment variable is not set"
    exit 1
fi
if test ! -d "${ITUAM_DATA}"
```

```
then
    echo "ERROR **** CS_pull: ${ITUAM_DATA} is not a directory"
    exit 1
fi

# test for presence of the ITUAM/UNIX environment configuration file
# ($ITUAM_DATA/A_config.par); this file enables various environment
# variables to be defined; if this file is missing, a step was missed
# during installation; this file should be created

if test ! -f "$ITUAM_DATA/A_config.par"
then
    echo "ERROR **** CS_pull: ITUAM/UNIX Configuration Parameter file
($ITUAM_DATA/A_config.par) not found"
    exit 1
fi

# test for ITUAM/UNIX etc directory which is the location of the scripts
# to set-up the ITUAM/UNIX platform and environment variables

: ${ITUAM_ETC:=`awk -F= '/^ITUAM_ETC/ {print $2}' $ITUAM_DATA/A_config.par`}

if test -z "${ITUAM_ETC}"
then
    echo "ERROR **** CS_pull: ITUAM_ETC environment variable is not set"
    exit 1
fi
if test ! -d "${ITUAM_ETC}"
then
    echo "ERROR **** CS_pull: ${ITUAM_ETC} is not a directory"
    exit 1
fi

if test -z "${ITUAM_ENVIRON}"
then
    . ${ITUAM_ETC}/ituam_env
fi

now=`date`
echo "
********************************************************************************
Starting ITUAM/UNIX CS_pull Script at ${now}
********************************************************************************
"
```

```
################################################################################
##
################################################################################
##
: ${CS_USER:=‘awk -F= ‘/^CS_USER[^_]/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${CS_KEY:=‘awk -F= ‘/^CS_KEY/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${CS_UPATH:=‘awk -F= ‘/^CS_UPATH/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${CS_METHOD:=‘awk -F= ‘/^CS_METHOD/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${CS_PROC_PATH:=‘awk -F= ‘/^CS_PROC_PATH/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${CS_PLATFORM:=‘awk -F= ‘/^CS_PLATFORM=/ {print $2}’ $ITUAM_DATA/A_config.par‘}

: ${ITUAM_USER:=‘awk -F= ‘/^ITUAM_USER/ {print $2}’ $ITUAM_DATA/A_config.par‘}

#
# Uncomment these lines and re-set these variables if using values other
#  than those in the ITUAM Environment Configuration File.
#
#CS_USER=
#CS_KEY=
#CS_UPATH=
################################################################################
##
################################################################################
##

#
# Now get the values of variables to determine which Input Source Files are to be
#  transferred.
#
################################################################################
##
################################################################################
##
: ${GEN_UNIXFS:=‘awk -F= ‘/^GEN_UNIXFS=/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${GEN_ORACLE:=‘awk -F= ‘/^GEN_ORACLE=/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${GEN_ORACLE_STORAGE:=‘awk -F= ‘/^GEN_ORACLE_STORAGE=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${GEN_DB2:=‘awk -F= ‘/^GEN_DB2=/ {print $2}’ $ITUAM_DATA/A_config.par‘}
: ${GEN_DB2_STORAGE:=‘awk -F= ‘/^GEN_DB2_STORAGE=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
################################################################################
##
################################################################################
##
```

```
#
# Now get the values of variables that define the processes sub-directory name for
each
#  input source file type.
#
#####################################################################################
##
#####################################################################################
##
: ${CS_UNIXOS_PROCDIR:=‘awk -F= ‘/^CS_UNIXOS_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${CS_UNIXFS_PROCDIR:=‘awk -F= ‘/^CS_UNIXFS_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${CS_ORA_PROCDIR:=‘awk -F= ‘/^CS_ORA_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${CS_ORA_STORAGE_PROCDIR:=‘awk -F= ‘/^CS_ORA_STORAGE_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${CS_DB2_PROCDIR:=‘awk -F= ‘/^CS_DB2_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
: ${CS_DB2_STORAGE_PROCDIR:=‘awk -F= ‘/^CS_DB2_STORAGE_PROCDIR=/ {print $2}’
$ITUAM_DATA/A_config.par‘}
#####################################################################################
##
#####################################################################################
##

#
# Define the sender platform name.
#
SENDER_PLATFORM=‘uname -n | awk -F. ‘{print $1}’‘

#
# Use internal renamed ITUAM variables to preclude exporting
# undesired changes
#
# @RC Hard code for now
ORIGIN=”/opt/ibm/tuam/collectors/Unix/CS_input_source”
XFER=”${CS_METHOD}”

if test “$XFER” != “FTP” -a “$XFER” != “SFTP” -a “$XFER” != “SCP” -a “$XFER” != “MV”
then
    $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4353E CS_METHOD
    if [ $? = 1 ]
    then
        echo “ERROR **** CS_pull: Invalid file transfer method set in CS_METHOD”
```

```
    fi
    exit 1
fi

####################################################################
# Need to do this on OSF1 TruClusters so that the value of {memb} is
#  correctly passed to the ftp call.
####################################################################
if test "$PLATFORM" = "OSF1"
then
    CURR_PWD=‘pwd‘
    cd $ORIGIN
    ORG_PWD=‘pwd‘
    ORIGIN=$ORG_PWD
    cd $CURR_PWD
fi
####################################################################

#
# Set default_file_list to the prefix of accounting files to be
#  transferred.
#

default_file_list="CS_sum_ "
if [ "${GEN_UNIXFS}" = "Y" ]
then
    default_file_list="$default_file_list CS_sum_fs_"
fi
if [ "${GEN_ORACLE}" = "Y" ]
then
    default_file_list="$default_file_list CS_sum_ora_"
fi
if [ "${GEN_ORACLE_STORAGE}" = "Y" ]
then
    default_file_list="$default_file_list CS_sum_orasto_"
fi
if [ "${GEN_DB2}" = "Y" ]
then
    default_file_list="$default_file_list CS_sum_db2_"
fi
if [ "${GEN_DB2_STORAGE}" = "Y" ]
then
    default_file_list="$default_file_list CS_sum_db2sto_"
fi
```

```
if test -z “$1”
then
    now=‘date +%Y%m%d‘
    DATE=‘echo “$now convert 0 1” | ${ITUAM_ETC}/ituam_date‘
    date_is_set=”Y”
else
for arg in $*
do
   case $arg in
    [12][0-9][0-9][0-9][01][0-9][0123][0-9])
     if test -n “${date_is_set}”
     then
       echo “there is already a date set, cannot use ${arg}”
     else
       DATE=${arg}
       date_is_set=”Y”
     fi
     ;;
    sum|SUM)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_”
     ;;
    unixfs|UNIXFS)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_fs_”
     ;;
    ora|ORA)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_ora_”
     ;;
    orasto|ORASTO)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_orasto_”
     ;;
    db2|DB2)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_db2_”
     ;;
    db2sto|DB2STO)
       built_files=”Y”
       temp_file_list=”${temp_file_list} CS_sum_db2sto_”
     ;;
    *)
     echo “ERROR: ***** CS_pull Invalid Argument ${arg} “
     exit 1
```

```
     ;;
  esac
done
fi

#
# Ensure that there is a date to use as a pattern
# Change this later to allow default date
#

if test -z "${date_is_set}"
then
    now=`date +%Y%m%d`
    DATE=`echo "$now convert 0 1" | ${ITUAM_ETC}/ituam_date`
    date_is_set="Y"
#   echo "Must have a date "
#   exit 1
fi

TEMP=`echo $DATE validate | ${ITUAM_ETC}/ituam_date`
if [ "$TEMP" = "0" ]
then
    $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4303E
    if [ $? = 1 ]
    then
        echo "ERROR **** CS_pull: Invalid date string."
    fi
    exit 1
fi

FILE_DATE=`echo $DATE convert -1 1 | ${ITUAM_ETC}/ituam_date`

#
#   Use default file list only if no files were individually specified
#

if test -z "${built_files}"
then
   file_list="${default_file_list}"
else
   file_list="${temp_file_list}"
fi

# echo "file_list = $file_list"
```

```
if test ! -d “${CS_UPATH}”
then
    CS_UPATH=“”
fi

#
# grep ORIGIN directory for files matching the pattern and date
# add them to a log file for processing
#

for arg in ${file_list}
do
    # echo “LOOKING for ${arg}${DATE}.csv”
    # @RC add 3 lines having deleted file existence check
    file_exists=‘ssh ${CS_USER}@${CS_PLATFORM} ls ${ORIGIN}/${arg}${DATE}.csv‘
    file_status=$?
    if test “$file_status” -gt “0”
    then
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4325W “${ORIGIN}/${arg}${DATE}.csv”
        if [ $? = 1 ]
        then
            echo “WARNING **** CS_pull: Files does not exist - ${arg}${DATE}.csv”
        fi
    fi
    log_files=”${log_files} ${arg}${DATE}.csv”
done

file_list=”${log_files}”

#
# CS_pull code follows
#

RETURN_CODE=0

if test “$CS_PLATFORM” = ““ -a “$CS_METHOD” != “MV”
then
    $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4300E CS_PLATFORM
    if [ $? = 1 ]
    then
        echo “ERROR **** CS_pull: CS_PLATFORM environment variable is not set”
    fi
    exit 1
fi
```

```
# @RC Files being retrieved from the remote server
echo "ITUAM/UNIX CS_pull: Retrieving files for $DATE from ${CS_PLATFORM} using $XFER"

#
# Transfer formatted ITUAM/UNIX Input Source Files to ITUAM Server Platform
#

if [ "${XFER}" = "FTP" ]
then

    #
    # Determine if on-the-fly ftp is required
    #

    if [ -n "${CS_USER}" -a -n "${CS_KEY}" -a -n "${CS_UPATH}" ]
    then
        ITUAM_FLY="Y"
        if test -r "${CS_UPATH}/.netrc"
        then
            mv  ${CS_UPATH}/.netrc  ${CS_UPATH}/.netrc_sav
            echo "ITUAM/UNIX CS_pull: ${CS_UPATH}/.netrc already exists. Moving
existing .netrc to .netrc_sav"
            if test -r "${CS_UPATH}/.netrc"
            then
                $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4307E ${CS_UPATH}/.netrc
${CS_UPATH}/.netrc_sav
                if [ $? = 1 ]
                then
                    echo "ERROR: Could not move ${CS_UPATH}/.netrc to
${CS_UPATH}/.netrc_sav"
                fi
                exit 1
            fi
        fi
    else
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4300E "CS_USER, CS_KEY or CS_UPATH"
        if [ $? = 1 ]
        then
            echo "ERROR **** CS_pull: CS_USER, CS_KEY or CS_UPATH environment
variable is not set"
        fi
        exit 1
    fi

    if test  "${ITUAM_FLY}" = "Y"
```

```
    then

MACHINE="${CS_PLATFORM}"
USER="${CS_USER}"
PASS="${CS_KEY}"

echo "
machine $MACHINE
login $USER
password $PASS

macdef sendb
binary
send \$1 \$2
" > ${CS_UPATH}/.netrc

       chmod 700 ${CS_UPATH}/.netrc
    fi

    for file_item in ${file_list}
    do

       define_destination $file_item

       ORIG_FILE=${ORIGIN}/${file_item}

       if test "$CS_PROC_PATH" = ""
       then
           FTP_DEST=${DESTIN}/${SENDER_PLATFORM}/${FILE_DATE}.txt
       else
           FTP_DEST=${CS_PROC_PATH}/${DESTIN}/${SENDER_PLATFORM}/${FILE_DATE}.txt
       fi

       echo "    Sending $ORIG_FILE to $FTP_DEST"

       #
       # Make sure the sub-folder exists
       #
       if test "$CS_PROC_PATH" = ""
       then
            echo "mkdir ${DESTIN}/${SENDER_PLATFORM}" \
                | ftp -v ${CS_PLATFORM}  1> /dev/null 2>&1
       else
            echo "mkdir ${CS_PROC_PATH}/${DESTIN}/${SENDER_PLATFORM}" \
                | ftp -v ${CS_PLATFORM}  1> /dev/null 2>&1
```

```
        fi

        #
        # Send the input source file.
        #
        echo "\$sendb ${ORIG_FILE} ${FTP_DEST}" \
             | ftp -v ${CS_PLATFORM}  1> ${ITUAM_HISTORY}/tmp_CS_pull_ftp_${DATE}.log
2>&1

        # Look for 226 at beginning of line. (FTP) Transfer complete status.
        #
        grep -s '^226 ' ${ITUAM_HISTORY}/tmp_CS_pull_ftp_${DATE}.log 2> /dev/null 1>&2
        status=$?
        if test "$status" -eq "0"
        then
            cat ${ITUAM_HISTORY}/tmp_CS_pull_ftp_${DATE}.log >>
${ITUAM_HISTORY}/CS_pull_ftp_${DATE}.log
            rm -f ${ITUAM_HISTORY}/tmp_CS_pull_ftp_${DATE}.log
        else
            mv ${ITUAM_HISTORY}/tmp_CS_pull_ftp_${DATE}.log
${ITUAM_HISTORY}/CS_pull_ftp_${DESTIN}_${DATE}.log
            $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4319E ${ORIG_FILE} ${FTP_DEST}
${XFER}
            if [ $? = 1 ]
            then
                echo "ERROR **** CS_pull: ${ORIG_FILE} not sent to ${FTP_DEST} using
${XFER}"
            fi

           RETURN_CODE=1
           echo " "
        fi
    done

    if test "${ITUAM_FLY}" = "Y"
    then
        rm -f ${CS_UPATH}/.netrc

        if test -f "${CS_UPATH}/.netrc_sav"
        then
            echo "ITUAM/UNIX CS_pull: Restoring original .netrc"
            mv ${CS_UPATH}/.netrc_sav ${CS_UPATH}/.netrc
        fi
     fi
```

```
elif [ "${XFER}" = "SCP" ]
then
    PATH="$PATH:/usr/local/bin"
    export PATH

    if [ -z "${CS_USER}" -o -z "${ITUAM_USER}" -o -z "${CS_PLATFORM}" -o -z
"${CS_PROC_PATH}" ]
    then
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4300E "CS_USER, CS_PLATFORM or
CS_PROC_PATH"
        if [ $? = 1 ]
        then
            echo "ERROR **** CS_pull: CS_USER, CS_PLATFORM or CS_PROC_PATH
environment variable is not set"
        fi
        exit 1
    fi

    for file_item in ${file_list}
    do
        define_destination $file_item
        ORIG_FILE=${ORIGIN}/${file_item}

        # @RC change line
        echo "    Retrieving $ORIG_FILE to
${CS_PROC_PATH}/${DESTIN}/${CS_PLATFORM}/${file_item}"

        #
        # Make sure the data source sub-folder exists
        # Remove check for existence of the output directory @RC

        #
        # Copy the source file to the ITUAM Server platform
        #
        # @RC replace the scp command to do a pull
        su - ${ITUAM_USER} -c "scp ${CS_USER}@${CS_PLATFORM}:${ORIG_FILE}
${CS_PROC_PATH}/${DESTIN}/${CS_PLATFORM}/${file_item}" 1>
${ITUAM_HISTORY}/tmp_CS_pull_${CS_PLATFORM}_scp_${DATE}.log 2>&1

        # Replace the output file with a more appropriate name @RC
        status=$?
        if test "$status" -eq "0"
        then
            cat ${ITUAM_HISTORY}/tmp_CS_pull_${CS_PLATFORM}_scp_${DATE}.log >>
${ITUAM_HISTORY}/CS_pull_${CS_PLATFORM}_scp_${DATE}.log
```

```
            echo "" >> ${ITUAM_HISTORY}/CS_pull_${CS_PLATFORM}_scp_${DATE}.log
            rm -f ${ITUAM_HISTORY}/tmp_CS_pull_${CS_PLATFORM}_scp_${DATE}.log
            echo " "
        else
            mv ${ITUAM_HISTORY}/tmp_CS_pull_${CS_PLATFORM}_scp_${DATE}.log
${ITUAM_HISTORY}/CS_pull_${CS_PLATFORM}_scp_${DESTIN}_${DATE}.log
            $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4319E ${ORIG_FILE} ${CS_PLATFORM}
${XFER}
            if [ $? = 1 ]
            then
                echo "ERROR **** CS_pull: ${ORIG_FILE} not sent to ${CS_PLATFORM}
using ${CS_METHOD}"
            fi
            RETURN_CODE=1
            echo " "
        fi
    done

elif [ "${XFER}" = "SFTP" ]
then
    PATH="$PATH:/usr/local/bin"
    export PATH

    #
    # Check the version of sftp to apply the correct option
    #
    SFTP_OPT="-b"
    SSH_VER='sftp -V 2>&1 | awk '{print $4}' | awk -F. '{print $1}''
    if test "$SSH_VER" = "3"
    then
        SFTP_OPT="-B"
    fi

    if [ -z "${CS_USER}" -o -z "${ITUAM_USER}" -o -z "${CS_PLATFORM}" ]
    then
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4300E "ITUAM_USER, CS_USER or
CS_PLATFORM"
        if [ $? = 1 ]
        then
            echo "ERROR **** CS_pull: ITUAM_USER, CS_USER or CS_PLATFORM environment
variable is not set"
        fi
        exit 1
    fi
```

```
    for file_item in ${file_list}
    do
        define_destination $file_item

        ORIG_FILE=${ORIGIN}/${file_item}

        if test “$CS_PROC_PATH” = ““
        then
            SFTP_DEST=”${DESTIN}/${SENDER_PLATFORM}/${FILE_DATE}.txt”
        else
            SFTP_DEST=”${CS_PROC_PATH}/${DESTIN}/${SENDER_PLATFORM}/${FILE_DATE}.txt”
        fi
        echo “    Sending $ORIG_FILE to $SFTP_DEST”

        #
        # Make sure the data source sub-folder exists
        #

        if test “$CS_PROC_PATH” = ““
        then
            echo “mkdir ${DESTIN}/${SENDER_PLATFORM}” > ${ORIGIN}/sftp_connect
        else
            echo “mkdir ${CS_PROC_PATH}/${DESTIN}/${SENDER_PLATFORM}” >
${ORIGIN}/sftp_connect
        fi

        su - ${ITUAM_USER} -c “sftp ${SFTP_OPT} ${ORIGIN}/sftp_connect
${CS_USER}@${CS_PLATFORM}” 2> /dev/null 1>&2
        rm ${ORIGIN}/sftp_connect

        #
        # Now build the ftp_connect file to transfer the input source
        #

        if test “$CS_PROC_PATH” = ““
        then
            echo “cd ${DESTIN}/${SENDER_PLATFORM}” > ${ORIGIN}/sftp_connect
        else
            echo “cd ${CS_PROC_PATH}/${DESTIN}/${SENDER_PLATFORM}” >
${ORIGIN}/sftp_connect
        fi

        echo “put ${ORIGIN}/${file_item} ${FILE_DATE}.txt” >> ${ORIGIN}/sftp_connect
```

```
        su - ${ITUAM_USER} -c “sftp ${SFTP_OPT} ${ORIGIN}/sftp_connect
${CS_USER}@${CS_PLATFORM}” 1> ${ITUAM_HISTORY}/tmp_CS_pull_sftp_${DATE}.log 2>&1

        grep -s “Uploading” ${ITUAM_HISTORY}/tmp_CS_pull_sftp_${DATE}.log 2>
/dev/null 1>&2
        status=$?
        if test “$status” -eq “0”
        then
            cat ${ITUAM_HISTORY}/tmp_CS_pull_sftp_${DATE}.log >>
${ITUAM_HISTORY}/CS_pull_sftp_${DATE}.log
            echo ““ >> ${ITUAM_HISTORY}/CS_pull_sftp_${DATE}.log
            rm -f ${ITUAM_HISTORY}/tmp_CS_pull_sftp_${DATE}.log
            echo “ “
        else
            mv ${ITUAM_HISTORY}/tmp_CS_pull_sftp_${DATE}.log
${ITUAM_HISTORY}/CS_pull_sftp_${DESTIN}_${DATE}.log
            $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4319E ${ORIGIN}/${file_item}
${CS_PLATFORM} ${CS_METHOD}
            if [ $? = 1 ]
            then
                echo “ERROR **** CS_pull: ${ORIGIN}/${file_item} not sent to
${CS_PLATFORM} using ${XFER}”
            fi
            RETURN_CODE=1
            echo “ “
        fi
    done

elif [ “${XFER}” = “MV” ]
then
    PATH=”$PATH:/usr/local/bin”
    export PATH

    if [ -z “${CS_PROC_PATH}” ]
    then
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4300E CS_PROC_PATH
        if [ $? = 1 ]
        then
            echo “ERROR **** CS_pull: CS_PROC_PATH environment variable is not set”
        fi
        exit 1
    fi

    if [ ! -d “${CS_PROC_PATH}” ]
    then
```

```
        $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4301E ${CS_PROC_PATH}
        if [ $? = 1 ]
        then
            echo "ERROR **** CS_pull: ${CS_PROC_PATH} is not a directory"
        fi
        exit 1
    fi

    for file_item in ${file_list}
    do
        define_destination $file_item
        ORIG_FILE=${ORIGIN}/${file_item}

        echo "ITUAM/UNIX CS_pull: Copying $ORIG_FILE to 
$CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM/$FILE_DATE.txt"

        #
        # Make sure the process sub-folder exists
        #

if [ ! -d $CS_PROC_PATH/$DESTIN ]
then
    mkdir $CS_PROC_PATH/$DESTIN
        fi

        #
        # Make sure the data source sub-folder exists
        #

if [ ! -d $CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM ]
then
    mkdir $CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM
        fi

        #
        # Move the source file to the Process Folder
        #

cp $ORIG_FILE $CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM/$FILE_DATE.txt
if [ ! -e $CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM/$FILE_DATE.txt ]
then
            $ITUAM_BIN/A_script_msg MESSAGE_AUCUC4308E $ORIG_FILE 
$CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM/$FILE_DATE.txt
            if [ $? = 1 ]
            then
```

```
                echo “ERROR **** CS_pull: Could not copy $ORIG_FILE to
$CS_PROC_PATH/$DESTIN/$SENDER_PLATFORM/$FILE_DATE.txt”
            fi
            RETURN_CODE=1
        fi
    done
fi

now=‘date‘
echo “
********************************************************************************
Ending ITUAM/UNIX CS_pull Script at ${now}
********************************************************************************
“
exit ${RETURN_CODE}
```

Sample job to load UNIX data

```
<?xml version=”1.0” encoding=”utf-8”?>
<!--
 ************************************************************ {COPYRIGHT-TOP}
 * Licensed Materials - Property of IBM
 * IBM Tivoli Usage and Accounting Manager
 * 5724-O33, 5765-UAV, 5765-UA7, 44E7863
 * (c) Copyright IBM Corp. 2004, 2007
 *
 * The source code for this program is not published or otherwise
 * divested of its trade secrets, irrespective of what has been
 * deposited with the U.S. Copyright Office.
 ************************************************************ {COPYRIGHT-END}
-->
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
  <Job   id=”LoadUNIX”
         description=”Job to process UNIX files sum and sum_fs data”
         active=”true”
         joblogShowStepParameters=”true”
         joblogShowStepOutput=”true”
         processPriorityClass=”Low”
         joblogWriteToTextFile=”true”
         joblogWriteToXMLFile=”true”
         stopOnProcessFailure=”false”>
    <Process   id=”UNIXProcessing”
               description=”Load UNIX sum and sum_fs data”
```

```
                active="true">
        <Steps stopOnStepFailure="true">

          <!-- ====================================== -->
          <!-- Step 1: Integrator with 3 stages        -->
          <!-- ====================================== -->
          <Step id="Integrator" type="ConvertToCSR" programType="integrator"
programName="integrator">
            <Integrator>
              <Input name="CSRInput" active="true">
                <Files>
                  <File
name="/opt/ibm/tuam/processes/UNIXProcessing/lpar04/CS_sum_%LogDate_End%.csv" />
                  <File
name="/opt/ibm/tuam/processes/UnixFS/lpar04/CS_sum_fs_%LogDate_End%.csv" />
                </Files>
              </Input>
              <!--  get account code fom table based on SYSTEM_ID (hostname -->
              <Stage name="CreateIdentifierFromTable" active="true">
                <Identifiers>
                  <Identifier name="Account_Code_TMP">
                    <FromIdentifiers>
                      <FromIdentifier name="SYSTEM_ID" offset="1" length="10"/>
                    </FromIdentifiers>
                  </Identifier>
                </Identifiers>
                <Files>
                  <File name="/opt/ibm/tuam/processes/Accttabl.txt" type="table"/>
                  <File name="UNIXException.txt" type="exception" format="CSROutput"/>
                </Files>
                <Parameters>
                  <Parameter exceptionProcess="true"/>
                  <Parameter sort="true"/>
                  <Parameter upperCase="false"/>
                  <Parameter writeNoMatch="false"/>
                  <Parameter modifyIfExists="true"/>
                </Parameters>
              </Stage>
              <!--  add hostname as last part to the account code -->
              <Stage name="CreateIdentifierFromIdentifiers" active="true">
                <Identifiers>
                  <Identifier name="Account_Code">
                    <FromIdentifiers>
                      <FromIdentifier name="Account_Code_TMP" offset="1" length="40"/>
                      <FromIdentifier name="SYSTEM_ID" offset="1" length="20"/>
```

```
          </FromIdentifiers>
        </Identifier>
      </Identifiers>
      <Parameters>
        <Parameter modifyIfExists="true"/>
        <Parameter keepLength="true"/>
      </Parameters>
    </Stage>
    <Stage name="CSRPlusOutput" active="true">
      <Files>
        <File name="AcctCSR.txt" />
      </Files>
    </Stage>
  </Integrator>
</Step>
<!-- ====================================== -->
<!-- Step 2: Process using program "Bill"    -->
<!-- ====================================== -->
<Step   id="Process"
        description="Standard Processing for UNIX"
        type="Process"
        programName="Bill"
        programType="java"
        active="true">
  <Bill>
    <Parameters>
    </Parameters>
  </Bill>
</Step>
<!-- ====================================== -->
<!-- Step 3: Process using program "DBLoad"  -->
<!-- ====================================== -->
<Step   id="DatabaseLoad"
        description="Database Load for UNIX"
        type="Process"
        programName="DBLoad"
        programType="java"
        active="true">
   <DBLoad>
     <Parameters>
     </Parameters>
   </DBLoad>
</Step>
<!-- ====================================== -->
<!-- Step 4: Process using program "Cleanup" -->
```

```
        <!-- ======================================= -->
        <Step   id=”Cleanup”
                description=”Cleanup UNIX”
                type=”Process”
                programName=”Cleanup”
                programType=”java”
                active=”false”>
          <Parameters>
            <Parameter DaysToRetainFiles=”45”/>
          </Parameters>
        </Step>
      </Steps>
    </Process>
  </Job>
</Jobs>
```

Sample job to load Windows process data

```
<?xml version=”1.0” encoding=”utf-8”?>
<!--

    ******************************************************** {COPYRIGHT-TOP}
    * Licensed Materials - Property of IBM
    * IBM Tivoli Usage and Accounting Manager
    * 5724-O33, 5765-UAV, 5765-UA7, 44E7863
    * (c) Copyright IBM Corp. 2004, 2007
    *
    * The source code for this program is not published or otherwise
    * divested of its trade secrets, irrespective of what has been
    * deposited with the U.S. Copyright Office.
    ******************************************************** {COPYRIGHT-END}
-->
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
   <Job id=”VBDProcess” description=”Daily collection” active=”true”
joblogShowStepParameters=”true” joblogShowStepOutput=”true”
processPriorityClass=”Low” joblogWriteToTextFile=”true” joblogWriteToXMLFile=”true”
smtpSendJobLog=”true” smtpServer=”mail.ITUAMCustomerCompany.com”
smtpFrom=”ITUAM@ITUAMCustomerCompany.com”
smtpTo=”John.ITUAMUser@ITUAMCustomerCompany.com” stopOnProcessFailure=”false”>
      <Process id=”VBDProcess” description=”Process for Windows Process Collection”
active=”true”>
         <Steps stopOnStepFailure=”true”>
```

```
          <Step id=”Integrator1” description=”Server1 WinProcess”
type=”ConvertToCSR” programName=”integrator” programType=”java” active=”true”>
            <Integrator>
              <Input active=”true” name=”CollectorInput”>
                <Collector name=”DELIMITED”>
                  <RecordDelimiter keyword=”NEWLINE”></RecordDelimiter>
                  <FieldDelimiter keyword=”TAB”></FieldDelimiter>
                  <TextFieldQualifier keyword=”NONE”></TextFieldQualifier>
                </Collector>
                <Parameters>
                  <Parameter name=”Header” value=”WinProc”></Parameter>
                  <Parameter name=”FeedName” value=”3C-000-C”></Parameter>
                </Parameters>
                <InputFields>
                  <InputField dataType=”STRING” name=”RecordType”
position=”1”></InputField>
                  <InputField dataType=”INTEGER” name=”ProcessId”
position=”2”></InputField>
                  <InputField dataType=”INTEGER” name=”ParentProcessId”
position=”3”></InputField>
                  <InputField dataType=”STRING” name=”ProcessName”
position=”4”></InputField>
                  <InputField dataType=”STRING” name=”ProcessPath”
position=”5”></InputField>
                  <InputField dataType=”STRING” name=”MachineName”
position=”6”></InputField>
                  <InputField dataType=”STRING” name=”UserName”
position=”7”></InputField>
                  <InputField dataType=”INTEGER”
name=”TerminalServicesSessionID” position=”8”></InputField>
                  <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”CreateDateTime” position=”9”></InputField>
                  <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”ExitDateTime” position=”10”></InputField>
                  <InputField dataType=”INTEGER” name=”ExitCode”
position=”11”></InputField>
                  <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”IntervalStartDateTime” position=”12”></InputField>
                  <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”IntervalEndDateTime” position=”13”></InputField>
                  <InputField dataType=”FLOAT” name=”ElapsedTimeSecs”
position=”14”></InputField>
                  <InputField dataType=”FLOAT” name=”CPUTimeSecs”
position=”15”></InputField>
```

```
                        <InputField dataType="FLOAT" name="KernelCPUTimeSecs"
position="16"></InputField>
                        <InputField dataType="FLOAT" name="UserCPUTimeSecs"
position="17"></InputField>
                        <InputField dataType="LONG" name="ReadRequests"
position="18"></InputField>
                        <InputField dataType="LONG" name="KBytesRead"
position="19"></InputField>
                        <InputField dataType="LONG" name="WriteRequests"
position="20"></InputField>
                        <InputField dataType="LONG" name="KBytesWritten"
position="21"></InputField>
                        <InputField dataType="LONG" name="PageFaultCount"
position="22"></InputField>
                        <InputField dataType="LONG" name="WorkingSetSizeKB"
position="23"></InputField>
                        <InputField dataType="LONG" name="PeakWorkingSetSizeKB"
position="24"></InputField>
                        <InputField dataType="LONG" name="PagefileUsageKB"
position="25"></InputField>
                        <InputField dataType="LONG" name="PeakPagefileUsageKB"
position="26"></InputField>
                        <InputField dataType="STRING" name="PriorityClass"
position="27"></InputField>
                        <InputField dataType="INTEGER" name="BasePriority"
position="28"></InputField>
                        <InputField dataType="INTEGER" name="SystemProcessorCount"
position="29"></InputField>
                        <InputField dataType="INTEGER" name="EligibleProcessorCount"
position="30"></InputField>
                        <InputField dataType="INTEGER" name="AffinityMask"
position="31"></InputField>
                        <InputField dataType="STRING" name="CIMSUProjectName"
position="32"></InputField>
                     </InputFields>
                     <OutputFields>
                        <OutputField name="headerrectype" src="PARAMETER"
srcName="Header"></OutputField>
                        <OutputField name="headerstartdate" src="INPUT"
srcName="CreateDateTime"></OutputField>
                        <OutputField name="headerenddate" src="INPUT"
srcName="CreateDateTime"></OutputField>
                        <OutputField name="headerstarttime" src="INPUT"
srcName="CreateDateTime"></OutputField>
```

```
                        <OutputField name=”headerendtime” src=”INPUT”
srcName=”CreateDateTime”></OutputField>
                        <OutputField name=”Feed” src=”PARAMETER”
srcName=”FeedName”></OutputField>
                        <OutputField name=”RecordType” src=”INPUT”
srcName=”RecordType”></OutputField>
                        <OutputField name=”ProcessId” src=”INPUT”
srcName=”ProcessId”></OutputField>
                        <OutputField name=”ParentProcessId” src=”INPUT”
srcName=”ParentProcessId”></OutputField>
                        <OutputField name=”ProcessName” src=”INPUT”
srcName=”ProcessName”></OutputField>
                        <OutputField name=”ProcessPath” src=”INPUT”
srcName=”ProcessPath”></OutputField>
                        <OutputField name=”Server” src=”INPUT”
srcName=”MachineName”></OutputField>
                        <OutputField name=”User” src=”INPUT”
srcName=”UserName”></OutputField>
                        <OutputField name=”PriorityClass” src=”INPUT”
srcName=”PriorityClass”></OutputField>
                        <OutputField name=”BasePriority” src=”INPUT”
srcName=”BasePriority”></OutputField>
                        <OutputField name=”WINELPTM” resource=”true” src=”INPUT”
srcName=”ElapsedTimeSecs”></OutputField>
                        <OutputField name=”WINCPUTM” resource=”true” src=”INPUT”
srcName=”CPUTimeSecs”></OutputField>
                        <OutputField name=”WINKCPUT” resource=”true” src=”INPUT”
srcName=”KernelCPUTimeSecs”></OutputField>
                        <OutputField name=”WINCPUUS” resource=”true” src=”INPUT”
srcName=”UserCPUTimeSecs”></OutputField>
                        <OutputField name=”WINRDREQ” resource=”true” src=”INPUT”
srcName=”ReadRequests”></OutputField>
                        <OutputField name=”WINKBYTR” resource=”true” src=”INPUT”
srcName=”KBytesRead”></OutputField>
                        <OutputField name=”WINWRREQ” resource=”true” src=”INPUT”
srcName=”WriteRequests”></OutputField>
                        <OutputField name=”WINKBWRI” resource=”true” src=”INPUT”
srcName=”KBytesWritten”></OutputField>
                        <OutputField name=”WINPGFLT” resource=”true” src=”INPUT”
srcName=”PageFaultCount”></OutputField>
                     </OutputFields>
                     <Files>
                        <File name=”C:\process.txt” type=”input”></File>
                     </Files>
                  </Input>
```

```
                  <Stage active="true" name="ExcludeRecsByValue">
                     <Identifiers>
                        <Identifier name="RecordType" cond="EQ"
value="RecordType"></Identifier>
                        <Identifier name="RecordType" cond="EQ"
value="E"></Identifier>
                        <Identifier name="RecordType" cond="EQ"
value="I"></Identifier>
                        <Identifier name="ParentProcessId" cond="EQ"
value="0"></Identifier>
                     </Identifiers>
                  </Stage>
                  <Stage active="true" name="DropFields">
                     <Fields>
                        <Field name="ParentProcessId"></Field>
                        <Field name="RecordType"></Field>
                     </Fields>
                  </Stage>
                  <Stage active="true" name="CSROutput">
                     <Files>
                        <File name="%ProcessFolder%\S\20071018.txt"></File>
                     </Files>
                  </Stage>
               </Integrator>
            </Step>
            <Step id="Integrator2" description="Server1 WinProcess"
type="ConvertToCSR" programName="integrator" programType="java" active="true">
               <Integrator>
                  <Input active="true" name="CollectorInput">
                     <Collector name="DELIMITED">
                        <RecordDelimiter keyword="NEWLINE"></RecordDelimiter>
                        <FieldDelimiter keyword="TAB"></FieldDelimiter>
                        <TextFieldQualifier keyword="NONE"></TextFieldQualifier>
                     </Collector>
                     <Parameters>
                        <Parameter name="Header" value="WinProc"></Parameter>
                        <Parameter name="FeedName" value="3C-000-C"></Parameter>
                     </Parameters>
                     <InputFields>
                        <InputField dataType="STRING" name="RecordType"
position="1"></InputField>
                        <InputField dataType="INTEGER" name="ProcessId"
position="2"></InputField>
                        <InputField dataType="INTEGER" name="ParentProcessId"
position="3"></InputField>
```

```
                        <InputField dataType=”STRING” name=”ProcessName”
position=”4”></InputField>
                        <InputField dataType=”STRING” name=”ProcessPath”
position=”5”></InputField>
                        <InputField dataType=”STRING” name=”MachineName”
position=”6”></InputField>
                        <InputField dataType=”STRING” name=”UserName”
position=”7”></InputField>
                        <InputField dataType=”INTEGER”
name=”TerminalServicesSessionID” position=”8”></InputField>
                        <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”CreateDateTime” position=”9”></InputField>
                        <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”ExitDateTime” position=”10”></InputField>
                        <InputField dataType=”INTEGER” name=”ExitCode”
position=”11”></InputField>
                        <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”IntervalStartDateTime” position=”12”></InputField>
                        <InputField dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss.hhh”
name=”IntervalEndDateTime” position=”13”></InputField>
                        <InputField dataType=”FLOAT” name=”ElapsedTimeSecs”
position=”14”></InputField>
                        <InputField dataType=”FLOAT” name=”CPUTimeSecs”
position=”15”></InputField>
                        <InputField dataType=”FLOAT” name=”KernelCPUTimeSecs”
position=”16”></InputField>
                        <InputField dataType=”FLOAT” name=”UserCPUTimeSecs”
position=”17”></InputField>
                        <InputField dataType=”LONG” name=”ReadRequests”
position=”18”></InputField>
                        <InputField dataType=”LONG” name=”KBytesRead”
position=”19”></InputField>
                        <InputField dataType=”LONG” name=”WriteRequests”
position=”20”></InputField>
                        <InputField dataType=”LONG” name=”KBytesWritten”
position=”21”></InputField>
                        <InputField dataType=”LONG” name=”PageFaultCount”
position=”22”></InputField>
                        <InputField dataType=”LONG” name=”WorkingSetSizeKB”
position=”23”></InputField>
                        <InputField dataType=”LONG” name=”PeakWorkingSetSizeKB”
position=”24”></InputField>
                        <InputField dataType=”LONG” name=”PagefileUsageKB”
position=”25”></InputField>
```

```
                       <InputField dataType=”LONG” name=”PeakPagefileUsageKB”
position=”26”></InputField>
                       <InputField dataType=”STRING” name=”PriorityClass”
position=”27”></InputField>
                       <InputField dataType=”INTEGER” name=”BasePriority”
position=”28”></InputField>
                       <InputField dataType=”INTEGER” name=”SystemProcessorCount”
position=”29”></InputField>
                       <InputField dataType=”INTEGER” name=”EligibleProcessorCount”
position=”30”></InputField>
                       <InputField dataType=”INTEGER” name=”AffinityMask”
position=”31”></InputField>
                       <InputField dataType=”STRING” name=”CIMSUProjectName”
position=”32”></InputField>
                    </InputFields>
                    <OutputFields>
                       <OutputField name=”headerrectype” src=”PARAMETER”
srcName=”Header”></OutputField>
                       <OutputField name=”headerstartdate” src=”INPUT”
srcName=”IntervalStartDateTime”></OutputField>
                       <OutputField name=”headerenddate” src=”INPUT”
srcName=”IntervalEndDateTime”></OutputField>
                       <OutputField name=”headerstarttime” src=”INPUT”
srcName=”IntervalStartDateTime”></OutputField>
                       <OutputField name=”headerendtime” src=”INPUT”
srcName=”IntervalEndDateTime”></OutputField>
                       <OutputField name=”Feed” src=”PARAMETER”
srcName=”FeedName”></OutputField>
                       <OutputField name=”RecordType” src=”INPUT”
srcName=”RecordType”></OutputField>
                       <OutputField name=”ProcessId” src=”INPUT”
srcName=”ProcessId”></OutputField>
                       <OutputField name=”ParentProcessId” src=”INPUT”
srcName=”ParentProcessId”></OutputField>
                       <OutputField name=”ProcessName” src=”INPUT”
srcName=”ProcessName”></OutputField>
                       <OutputField name=”ProcessPath” src=”INPUT”
srcName=”ProcessPath”></OutputField>
                       <OutputField name=”Server” src=”INPUT”
srcName=”MachineName”></OutputField>
                       <OutputField name=”User” src=”INPUT”
srcName=”UserName”></OutputField>
                       <OutputField name=”PriorityClass” src=”INPUT”
srcName=”PriorityClass”></OutputField>
```

```
                       <OutputField name=”BasePriority” src=”INPUT”
srcName=”BasePriority”></OutputField>
                       <OutputField name=”WINELPTM” resource=”true” src=”INPUT”
srcName=”ElapsedTimeSecs”></OutputField>
                       <OutputField name=”WINCPUTM” resource=”true” src=”INPUT”
srcName=”CPUTimeSecs”></OutputField>
                       <OutputField name=”WINKCPUT” resource=”true” src=”INPUT”
srcName=”KernelCPUTimeSecs”></OutputField>
                       <OutputField name=”WINCPUUS” resource=”true” src=”INPUT”
srcName=”UserCPUTimeSecs”></OutputField>
                       <OutputField name=”WINRDREQ” resource=”true” src=”INPUT”
srcName=”ReadRequests”></OutputField>
                       <OutputField name=”WINKBYTR” resource=”true” src=”INPUT”
srcName=”KBytesRead”></OutputField>
                       <OutputField name=”WINWRREQ” resource=”true” src=”INPUT”
srcName=”WriteRequests”></OutputField>
                       <OutputField name=”WINKBWRI” resource=”true” src=”INPUT”
srcName=”KBytesWritten”></OutputField>
                       <OutputField name=”WINPGFLT” resource=”true” src=”INPUT”
srcName=”PageFaultCount”></OutputField>
                    </OutputFields>
                    <Files>
                       <File name=”C:\process.txt” type=”input”></File>
                    </Files>
                 </Input>
                 <Stage active=”true” name=”ExcludeRecsByValue”>
                    <Identifiers>
                       <Identifier name=”RecordType” cond=”EQ”
value=”RecordType”></Identifier>
                       <Identifier name=”RecordType” cond=”EQ”
value=”E”></Identifier>
                       <Identifier name=”RecordType” cond=”EQ”
value=”S”></Identifier>
                       <Identifier name=”ParentProcessId” cond=”EQ”
value=”0”></Identifier>
                    </Identifiers>
                 </Stage>
                 <Stage active=”true” name=”DropFields”>
                    <Fields>
                       <Field name=”ParentProcessId”></Field>
                       <Field name=”RecordType”></Field>
                    </Fields>
                 </Stage>
                 <Stage active=”true” name=”CSRPlusOutput”>
                    <Files>
```

```
                        <File name="%ProcessFolder%\I\20071018.txt"></File>
                     </Files>
                  </Stage>
               </Integrator>
            </Step>
            <Step id="Scan" description="Scan LoadWinProcess" type="Process"
programName="Scan" programType="java" active="true">
               <Parameters>
                  <Parameter retainFileDate="false"></Parameter>
                  <Parameter allowMissingFiles="true"></Parameter>
                  <Parameter allowEmptyFiles="true"></Parameter>
                  <Parameter useStepFiles="false"></Parameter>
               </Parameters>
            </Step>
            <Step id="Integrator3" description="Server1 WinProcess" type="Process"
programName="integrator" programType="java" active="true">
               <Integrator>
                  <Input name="CSRInput" active="true">
                     <Files>
                        <File name="%ProcessFolder%/CurrentCSR.txt"></File>
                     </Files>
                  </Input>
                  <Stage name="Aggregator" active="true" trace="false">
                     <Identifiers>
                        <Identifier name="Feed"></Identifier>
                        <Identifier name="ProcessName"></Identifier>
                        <Identifier name="ProcessPath"></Identifier>
                        <Identifier name="Server"></Identifier>
                        <Identifier name="User"></Identifier>
                        <Identifier name="PriorityClass"></Identifier>
                        <Identifier name="BasePriority"></Identifier>
                     </Identifiers>
                     <Resources>
                        <Resource name="WINELPTM"></Resource>
                        <Resource name="WINCPUTM"></Resource>
                        <Resource name="WINKCPUT"></Resource>
                        <Resource name="WINCPUUS"></Resource>
                        <Resource name="WINRDREQ"></Resource>
                        <Resource name="WINKBYTR"></Resource>
                        <Resource name="WINWRREQ"></Resource>
                        <Resource name="WINKBWRI"></Resource>
                     </Resources>
                     <Parameters>
                        <Parameter defaultAggregation="false"></Parameter>
                     </Parameters>
```

```
                  </Stage>
                  <Stage name=”CreateIdentifierFromIdentifiers” active=”true”
trace=”false” stopOnStageFailure=”true”>
                     <Identifiers>
                        <Identifier name=”Account_Code”>
                           <FromIdentifiers>
                              <FromIdentifier name=”Server” offset=”1”
length=”12”></FromIdentifier>
                              <FromIdentifier name=”User” offset=”1”
length=”12”></FromIdentifier>
                           </FromIdentifiers>
                        </Identifier>
                     </Identifiers>
                     <Parameters>
                        <Parameter keepLength=”true”></Parameter>
                        <Parameter modifyIfExists=”true”></Parameter>
                     </Parameters>
                  </Stage>
                  <Stage name=”CSRPlusOutput” active=”true”>
                     <Files>
                        <File name=”%ProcessFolder%/AcctCSR.txt”></File>
                     </Files>
                  </Stage>
               </Integrator>
            </Step>
         </Steps>
      </Process>
   </Job>
</Jobs>
```

Sample job for Tivoli Decision Support for z/OS

```
<?xml version=”1.0” encoding=”utf-8”?>
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
    <Job    id=”Sample-TDSz”
            description=”Daily TDSz collection”
            active=”true”
            joblogWriteToDB=”false”
            joblogWriteToTextFile=”true”
            joblogWriteToXMLFile=”true”
            joblogShowStepOutput=”true”
            joblogShowStepParameters=”true”
            processPriorityClass=”Low”
```

```
            smtpServer="mail.ITUAMCustomerCompany.com"
            smtpFrom="ITUAM@ITUAMCustomerCompany.com"
            smtpTo="John.ITUAMUser@ITUAMCustomerCompany.com"
            stopOnProcessFailure="false">
        <Process    id="IBMTDS"
                    description="Process for TDS Collection"
                    joblogShowStepOutput="true"
                    joblogShowStepParameters="true"
                    active="true">
            <Steps stopOnStepFailure="true">
                <Step id="Integrator-TDS-RAFADDRLOG" type="ConvertToCSR"
programType="java" programName="integrator"
                    active="true">
                    <Integrator>
                        <Input name="CollectorInput" active="true">

                            <Collector name ="TDS">
                                <Connection dataSourceName="DB8D"/>
                                            <Statement text="SELECT  a_timestamp as
date, decimal(a_srbtime, 20, 6) as srbtime, decimal(a_tcbtime, 20, 6) as tcbtime,
decimal(a_ndiskblks) as diskblks, decimal(a_ntapeblks) as tapeblks, decimal(a_nexcps)
as excps, a_stcname as stcname, a_smfid as sysid, a_acct1 as acct1,a_acct2 as acct2,
a_acct3 as acct3,  a_acct4 as acct4, a_acct5 as acct5, a_pgmname as programnm,
a_userid as userid
                                FROM DRL.rafaddrlog Where a_timestamp &gt;= ? and
a_timestamp &lt;= ?"/>
                                <Parameter src="PARAMETER" sqlType="TIMESTAMP"
position="1" srcName="StartLogDate"/>
                                <Parameter src="PARAMETER" sqlType="TIMESTAMP"
position="2" srcName="EndLogDate"/>
                            </Collector>

                            <Parameters>
                               <Parameter name="StartLogDate" value="%LogDate_Start%
00:00:00" dataType="DATETIME" format="yyyyMMdd HH:mm:ss"/>
                                <Parameter name="EndLogDate" value="%LogDate_End%
23:59:59" dataType="DATETIME" format="yyyyMMdd HH:mm:ss"/>
                                <Parameter name="Resourceheader" value="TDSzADRL"
dataType="STRING"/>
                                <Parameter name="Feed" value="server1"
dataType="STRING"/>
                                <Parameter name="LogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                            </Parameters>
```

```
                        <InputFields>
                            <InputField name="1" columnName="DATE"
dataType="DATETIME" format="yyyyMMdd"/>
                            <InputField name="2" columnName="SRBTIME"
dataType="DOUBLE"/>
                            <InputField name="3" columnName="TCBTIME"
dataType="DOUBLE"/>
                            <InputField name="4" columnName="DISKBLKS"
dataType="DOUBLE"/>
                            <InputField name="5" columnName="TAPEBLKS"
dataType="DOUBLE"/>
                            <InputField name="6" columnName="EXCPS"
dataType="DOUBLE"/>
                            <InputField name="7" columnName="STCNAME"
dataType="STRING"/>
                            <InputField name="8" columnName="SYSID"
dataType="STRING"/>
                            <InputField name="9" columnName="ACCT1"
dataType="STRING"/>
                            <InputField name="10" columnName="ACCT2"
dataType="STRING"/>
                            <InputField name="11" columnName="ACCT3"
dataType="STRING"/>
                            <InputField name="12" columnName="ACCT4"
dataType="STRING"/>
                            <InputField name="13" columnName="ACCT5"
dataType="STRING"/>
                            <InputField name="14" columnName="PROGRAMNM"
dataType="STRING"/>
                            <InputField name="15" columnName="USERID"
dataType="STRING"/>
                        </InputFields>

                        <OutputFields>
                            <OutputField name="Feed" src="PARAMETER"
srcName="Feed" />
                            <OutputField name="headerstartdate" src="INPUT"
srcName="1" />
                            <OutputField name="headerenddate" src="INPUT"
srcName="1" />
                            <OutputField name="STCname" src="INPUT" srcName="7"
/>
                            <OutputField name="SYSID" src="INPUT" srcName="8" />
                            <OutputField name="ACCT1" src="INPUT" srcName="9" />
                            <OutputField name="ACCT2" src="INPUT" srcName="10" />
```

```
                                <OutputField name="ACCT3" src="INPUT" srcName="11" />
                                <OutputField name="ACCT4" src="INPUT" srcName="12" />
                                <OutputField name="ACCT5" src="INPUT" srcName="13" />
                                <OutputField name="PROGRAMNM" src="INPUT"
srcName="14" />
                                <OutputField name="USERID" src="INPUT" srcName="15"
/>
                                <OutputField name="TALSRBT" src="INPUT" srcName="2"
resource="true" />
                                <OutputField name="TALTCBT" src="INPUT" srcName="3"
resource="true" />
                                <OutputField name="TALDBLK" src="INPUT" srcName="4"
resource="true" />
                                <OutputField name="TALTBLK" src="INPUT" srcName="5"
resource="true" />
                                <OutputField name="TALEXCP" src="INPUT" srcName="6"
resource="true" />
                            </OutputFields>

                            <Files>
                                <File name="%ProcessFolder%/exceptionADRL.txt"
type="exception" />
                            </Files>

                        </Input>
                        <Stage name="CreateIdentifierFromIdentifiers" active="true"
trace="false" stopOnStageFailure="true" >
                            <Identifiers>
                               <Identifier name="Account_Code">
                                   <FromIdentifiers>
                         <FromIdentifier name="SYSID" offset="1" length="6"/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength="true"/>
                               <Parameter modifyIfExists="true"/>
                            </Parameters>
                        </Stage>
                        <Stage name="CSRPlusOutput" active="true">
                            <Files>
                                <File
name="%ProcessFolder%/server1/%LogDate_End%-TDSzADRL.txt"  />
                            </Files>
                        </Stage>
```

```
                </Integrator>
            </Step>
            <Step id="Integrator-TDS-RAFBATCH" type="ConvertToCSR"
programType="java" programName="integrator"
                active="true">
                <Integrator>
                    <Input name="CollectorInput" active="true">

                        <Collector name ="TDS">
                            <Connection dataSourceName="DB8D"/>
                                        <Statement text="SELECT
decimal(cpusec,20,6) as cpu, decimal(njobs) as jobs,
                            decimal(npages) as pages, decimal(prtlines) as
prtlines,
                            decimal(srbtime, 20, 6) as srbtime, decimal(tcbtime,
20, 6) as tcbtime,
                            decimal(excps) as excps, decimal(diskblks) as
diskblks,
                            decimal(tapeblks) as tapeblks,
                            jobname, period, sysid, account, date, printer
                            FROM drl.rafbatch Where date &gt;= ? and date &lt;=
?"/>
                            <Parameter src="PARAMETER" sqlType="DATE"
position="1" srcName="StartLogDate"/>
                            <Parameter src="PARAMETER" sqlType="DATE"
position="2" srcName="EndLogDate"/>
                        </Collector>

                        <Parameters>
                            <Parameter name="StartLogDate"
value="%LogDate_Start%" dataType="DATETIME" format="yyyyMMdd"/>
                            <Parameter name="EndLogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                            <Parameter name="Resourceheader" Value="TDSzBAT"
dataType="STRING"/>
                            <Parameter name="Feed" value="server1"
dataType="STRING"/>
                            <Parameter name="LogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                        </Parameters>

                        <InputFields>
                            <InputField name="1" columnName="CPU"
dataType="STRING"/>
```

```
                                <InputField name="2" columnName="JOBS"
dataType="STRING"/>
                                <InputField name="3" columnName="PAGES"
dataType="STRING"/>
                                <InputField name="4" columnName="PRTLINES"
dataType="STRING"/>
                                <InputField name="5" columnName="SRBTIME"
dataType="STRING"/>
                                <InputField name="6" columnName="TCBTIME"
dataType="STRING"/>
                                <InputField name="7" columnName="EXCPS"
dataType="STRING"/>
                                <InputField name="8" columnName="DISKBLKS"
dataType="STRING"/>
                                <InputField name="9" columnName="TAPEBLKS"
dataType="STRING"/>
                                <InputField name="10" columnName="JOBNAME"
dataType="STRING"/>
                                <InputField name="11" columnName="PERIOD"
dataType="STRING"/>
                                <InputField name="12" columnName="SYSID"
dataType="STRING"/>
                                <InputField name="13" columnName="ACCOUNT"
dataType="STRING"/>
                                <InputField name="14" columnName="DATE"
dataType="DATETIME" format="yyyyMMdd"/>
                                <InputField name="15" columnName="PRINTER"
dataType="STRING"/>
                            </InputFields>

                            <OutputFields>
                                <OutputField name="Feed" src="PARAMETER"
srcName="Feed" />
                                <OutputField name="headerstartdate" src="INPUT"
srcName="14" />
                                <OutputField name="headerenddate" src="INPUT"
srcName="14" />
                                <OutputField name="ACCOUNT" src="INPUT" srcName="13"
/>
                                <OutputField name="PERIOD" src="INPUT" srcName="11"
/>
                                <OutputField name="DATE" src="INPUT" srcName="14" />
                                <OutputField name="JOBNAME" src="INPUT" srcName="10"
/>
```

```
                                <OutputField name=”PRINTER” src=”INPUT” srcName=”15”
/>
                                <OutputField name=”Z003” src=”INPUT” srcName=”1”
resource=”true” />
                                <OutputField name=”Z001” src=”INPUT” srcName=”2”
resource=”true” />
                                <OutputField name=”Z017” src=”INPUT” srcName=”3”
resource=”true” />
                                <OutputField name=”Z016” src=”INPUT” srcName=”4”
resource=”true” />
                                <OutputField name=”TBASRBT” src=”INPUT” srcName=”5”
resource=”true” />
                                <OutputField name=”TBATCBT” src=”INPUT” srcName=”6”
resource=”true” />
                                <OutputField name=”Z006” src=”INPUT” srcName=”8”
resource=”true” />
                                <OutputField name=”Z007” src=”INPUT” srcName=”9”
resource=”true” />
                                <OutputField name=”TBAEXCP” src=”INPUT” srcName=”7”
resource=”true” />
                            </OutputFields>

                            <Files>
                                <File name=”%ProcessFolder%/exceptionBAT.txt”
type=”exception” />
                            </Files>

                        </Input>
                        <Stage name=”CreateIdentifierFromIdentifiers” active=”true”
trace=”false” stopOnStageFailure=”true” >
                            <Identifiers>
                               <Identifier name=”Account_Code”>
                                   <FromIdentifiers>
                         <FromIdentifier name=”ACCOUNT” offset=”1” length=”6”/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength=”true”/>
                               <Parameter modifyIfExists=”true”/>
                            </Parameters>
                        </Stage>
                        <Stage name=”CSRPlusOutput” active=”true”>
                            <Files>
```

```
                                <File
name=”%ProcessFolder%/server1/%LogDate_End%-TDSzBAT.txt”  />
                            </Files>
                        </Stage>
                    </Integrator>
                </Step>
                <Step id=”Integrator-TDS-RAFJOBLOG” type=”ConvertToCSR”
programType=”java” programName=”integrator”
                    active=”true”>
                    <Integrator>
                        <Input name=”CollectorInput” active=”true”>

                            <Collector name =”TDS”>
                                <Connection dataSourceName=”DB8D”/>
                                            <Statement text=”SELECT  j_timestamp as
date, decimal(j_srbtime, 20, 6) as srbtime,
                                    decimal(j_tcbtime, 20, 6) as tcbtime,
decimal(j_mdisks) as mdisks,
                                    decimal(j_mtapes) as mtapes, decimal(j_nlines) as
nlines,
                                    decimal(j_wtrlines) as wtrlines, j_jobname as
jobname, j_smfid as sysid,
                                    j_acct1 as acct1,j_acct2 as acct2, j_acct3 as
acct3,  j_acct4 as acct4,
                                    j_acct5 as acct5, j_printer as printer,
j_pgmrname as programmer,
                                    j_userid as userid
                                    FROM drl.rafjoblog Where j_timestamp &gt;= ? and
j_timestamp &lt;= ?”/>
                                <Parameter src=”PARAMETER” sqlType=”TIMESTAMP”
position=”1” srcName=”StartLogDate”/>
                                <Parameter src=”PARAMETER” sqlType=”TIMESTAMP”
position=”2” srcName=”EndLogDate”/>
                            </Collector>

                            <Parameters>
                               <Parameter name=”StartLogDate” value=”%LogDate_Start%
00:00:00” dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss”/>
                                <Parameter name=”EndLogDate” value=”%LogDate_End%
23:59:59” dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss”/>
                                <Parameter name=”Resourceheader” Value=”TDSzJOBL”
dataType=”STRING”/>
                                <Parameter name=”Feed” value=”server1”
dataType=”STRING”/>
```

```
                            <Parameter name=”LogDate” value=”%LogDate_End%”
dataType=”DATETIME” format=”yyyyMMdd”/>
                        </Parameters>

                        <InputFields>
                            <InputField name=”1” columnName=”DATE”
dataType=”DATETIME” format=”yyyyMMdd”/>
                            <InputField name=”2” columnName=”SRBTIME”
dataType=”STRING”/>
                            <InputField name=”3” columnName=”TCBTIME”
dataType=”STRING”/>
                            <InputField name=”4” columnName=”MDISKS”
dataType=”STRING”/>
                            <InputField name=”5” columnName=”MTAPES”
dataType=”STRING”/>
                            <InputField name=”6” columnName=”NLINES”
dataType=”STRING”/>
                            <InputField name=”7” columnName=”WTRLINES”
dataType=”STRING”/>
                            <InputField name=”8” columnName=”JOBNAME”
dataType=”STRING”/>
                            <InputField name=”9” columnName=”SYSID”
dataType=”STRING”/>
                            <InputField name=”10” columnName=”ACCT1”
dataType=”STRING”/>
                            <InputField name=”11” columnName=”ACCT2”
dataType=”STRING”/>
                            <InputField name=”12” columnName=”ACCT3”
dataType=”STRING”/>
                            <InputField name=”13” columnName=”ACCT4”
dataType=”STRING”/>
                            <InputField name=”14” columnName=”ACCT5”
dataType=”STRING”/>
                            <InputField name=”15” columnName=”PRINTER”
dataType=”STRING”/>
                            <InputField name=”16” columnName=”PROGRAMMER”
dataType=”STRING”/>
                            <InputField name=”17” columnName=”USERID”
dataType=”STRING”/>
                        </InputFields>

                        <OutputFields>

                            <OutputField name=”Feed” src=”PARAMETER”
srcName=”Feed” />
```

```
                              <OutputField name="headerstartdate" src="INPUT"
srcName="1" />
                              <OutputField name="headerenddate" src="INPUT"
srcName="1" />
                              <OutputField name="JOBNAME" src="INPUT" srcName="8"
/>
                              <OutputField name="SYSID" src="INPUT" srcName="9" />
                              <OutputField name="ACCT1" src="INPUT" srcName="10" />
                              <OutputField name="ACCT2" src="INPUT" srcName="11" />
                              <OutputField name="ACCT3" src="INPUT" srcName="12" />
                              <OutputField name="ACCT4" src="INPUT" srcName="13" />
                              <OutputField name="ACCT5" src="INPUT" srcName="14" />
                              <OutputField name="PRINTER" src="INPUT" srcName="15"
/>
                              <OutputField name="PROGRAMMER" src="INPUT"
srcName="16" />
                              <OutputField name="USERID" src="INPUT" srcName="17"
/>
                              <OutputField name="TJLSRBT" src="INPUT" srcName="2"
resource="true" />
                              <OutputField name="TJLTCBT" src="INPUT" srcName="3"
resource="true" />
                              <OutputField name="TJLMDISK" src="INPUT" srcName="4"
resource="true" />
                              <OutputField name="TJLMTAPE" src="INPUT" srcName="5"
resource="true" />
                              <OutputField name="TJLNLINE" src="INPUT" srcName="6"
resource="true" />
                              <OutputField name="TJLWTRL" src="INPUT" srcName="7"
resource="true" />
                          </OutputFields>

                          <Files>
                              <File name="%ProcessFolder%/exceptionJOBL.txt"
type="exception" />
                          </Files>

                      </Input>
                      <Stage name="CreateIdentifierFromIdentifiers" active="true"
trace="false" stopOnStageFailure="true" >
                          <Identifiers>
                             <Identifier name="Account_Code">
                                 <FromIdentifiers>
                       <FromIdentifier name="SYSID" offset="1" length="6"/>
                                 </FromIdentifiers>
```

```
                              </Identifier>
                           </Identifiers>
                           <Parameters>
                              <Parameter keepLength=”true”/>
                              <Parameter modifyIfExists=”true”/>
                           </Parameters>
                       </Stage>
                       <Stage name=”CSRPlusOutput” active=”true”>
                           <Files>
                               <File 
name=”%ProcessFolder%/server1/%LogDate_End%-TDSzJOBL.txt”  />
                           </Files>
                       </Stage>
                   </Integrator>
               </Step>
               <Step id=”Integrator-TDS-RAFSESL” type=”ConvertToCSR” 
programType=”java” programName=”integrator”
                   active=”true”>
                   <Integrator>
                       <Input name=”CollectorInput” active=”true”>

                           <Collector name =”TDS”>
                               <Connection dataSourceName=”DB8D”/>
                                           <Statement text=”SELECT  s_timestamp as 
date, decimal(s_srbtime, 20, 6) as srbtime, 
                                   decimal(s_tcbtime, 20, 6) as tcbtime, 
decimal(s_nexcps) as nexcps, 
                                   decimal(s_rcttime,20,6) as rcttime, s_jobname as 
jobname, s_smfid as sysid, 
                                   s_acct1 as acct1,s_acct2 as acct2, s_acct3 as 
acct3,  s_acct4 as acct4, 
                                   s_acct5 as acct5, s_termid as termid,s_userid as 
userid 
                                   FROM drl.rafseslog Where s_timestamp &gt;= ? and 
s_timestamp &lt;= ?”/>
                               <Parameter src=”PARAMETER” sqlType=”TIMESTAMP” 
position=”1” srcName=”StartLogDate”/>
                               <Parameter src=”PARAMETER” sqlType=”TIMESTAMP” 
position=”2” srcName=”EndLogDate”/>
                           </Collector>

                           <Parameters>
                              <Parameter name=”StartLogDate” value=”%LogDate_Start% 
00:00:00” dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss”/>
```

```
                              <Parameter name=”EndLogDate” value=”%LogDate_End%
23:59:59” dataType=”DATETIME” format=”yyyyMMdd HH:mm:ss”/>
                              <Parameter name=”Resourceheader” Value=”TDSzSESL”
dataType=”STRING”/>
                              <Parameter name=”Feed” value=”server1”
dataType=”STRING”/>
                              <Parameter name=”LogDate” value=”%LogDate_End%”
dataType=”DATETIME” format=”yyyyMMdd”/>
                          </Parameters>

                          <InputFields>
                              <InputField name=”1” columnName=”DATE”
dataType=”DATETIME” format=”yyyyMMdd”/>
                              <InputField name=”2” columnName=”SRBTIME”
dataType=”STRING”/>
                              <InputField name=”3” columnName=”TCBTIME”
dataType=”STRING”/>
                              <InputField name=”4” columnName=”NEXCPS”
dataType=”STRING”/>
                              <InputField name=”5” columnName=”RCTTIME”
dataType=”STRING”/>
                              <InputField name=”6” columnName=”JOBNAME”
dataType=”STRING”/>
                              <InputField name=”7” columnName=”SYSID”
dataType=”STRING”/>
                              <InputField name=”8” columnName=”ACCT1”
dataType=”STRING”/>
                              <InputField name=”9” columnName=”ACCT2”
dataType=”STRING”/>
                              <InputField name=”10” columnName=”ACCT3”
dataType=”STRING”/>
                              <InputField name=”11” columnName=”ACCT4”
dataType=”STRING”/>
                              <InputField name=”12” columnName=”ACCT5”
dataType=”STRING”/>
                              <InputField name=”13” columnName=”TERMID”
dataType=”STRING”/>
                              <InputField name=”14” columnName=”USERID”
dataType=”STRING”/>
                          </InputFields>

                          <OutputFields>

                              <OutputField name=”Feed” src=”PARAMETER”
srcName=”Feed” />
```

```
                                <OutputField name="headerstartdate" src="INPUT"
srcName="1" />
                                <OutputField name="headerenddate" src="INPUT"
srcName="1" />
                                <OutputField name="JOBNAME" src="INPUT" srcName="6"
/>
                                <OutputField name="SYSID" src="INPUT" srcName="7" />
                                <OutputField name="ACCT1" src="INPUT" srcName="8" />
                                <OutputField name="ACCT2" src="INPUT" srcName="9" />
                                <OutputField name="ACCT3" src="INPUT" srcName="10" />
                                <OutputField name="ACCT4" src="INPUT" srcName="11" />
                                <OutputField name="ACCT5" src="INPUT" srcName="12" />
                                <OutputField name="TERMID" src="INPUT" srcName="13"
/>
                                <OutputField name="USERID" src="INPUT" srcName="14"
/>
                                <OutputField name="TSLSRBT" src="INPUT" srcName="2"
resource="true" />
                                <OutputField name="TSLTCBT" src="INPUT" srcName="3"
resource="true" />
                                <OutputField name="TSLEXCPS" src="INPUT" srcName="4"
resource="true" />
                                <OutputField name="TSLRCTME" src="INPUT" srcName="5"
resource="true" />
                            </OutputFields>

                            <Files>
                                <File name="%ProcessFolder%/exceptionSESL.txt"
type="exception" />
                            </Files>

                        </Input>
                        <Stage name="CreateIdentifierFromIdentifiers" active="true"
trace="false" stopOnStageFailure="true" >
                            <Identifiers>
                               <Identifier name="Account_Code">
                                   <FromIdentifiers>
                         <FromIdentifier name="SYSID" offset="1" length="6"/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength="true"/>
                               <Parameter modifyIfExists="true"/>
                            </Parameters>
```

```
                        </Stage>
                        <Stage name="CSRPlusOutput" active="true">
                            <Files>
                                <File
name="%ProcessFolder%/server1/%LogDate_End%-TDSzSESL.txt"  />
                            </Files>
                        </Stage>
                    </Integrator>
                </Step>
                <Step id="Integrator-TDS-RAFSTC" type="ConvertToCSR"
programType="java" programName="integrator"
                    active="true">
                    <Integrator>
                        <Input name="CollectorInput" active="true">

                            <Collector name ="TDS">
                                <Connection dataSourceName="DB8D"/>
                                            <Statement text="SELECT
decimal(cpusec,20,6) as cpu, decimal(nstcs) as nstcs,
                                    decimal(excps) as excps, decimal(diskblks) as
diskblks,
                                    decimal(tapeblks) as tapeblks,
                                    decimal(srbtime, 20, 6) as srbtime,
decimal(tcbtime, 20, 6) as tcbtime,
                                    stcname, period, sysid, account, date
                                    FROM drl.rafstc Where date &gt;= ? and date &lt;=
?"/>
                                <Parameter src="PARAMETER" sqlType="DATE"
position="1" srcName="StartLogDate"/>
                                <Parameter src="PARAMETER" sqlType="DATE"
position="2" srcName="EndLogDate"/>
                            </Collector>

                            <Parameters>
                                <Parameter name="StartLogDate"
value="%LogDate_Start%" dataType="DATETIME" format="yyyyMMdd"/>
                                <Parameter name="EndLogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                                <Parameter name="Resourceheader" Value="TDSzSTC"
dataType="STRING"/>
                                <Parameter name="Feed" value="server1"
dataType="STRING"/>
                                <Parameter name="LogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                            </Parameters>
```

```
                            <InputFields>
                                <InputField name=”1” columnName=”CPU”
dataType=”STRING”/>
                                <InputField name=”2” columnName=”NSTCS”
dataType=”STRING”/>
                                <InputField name=”3” columnName=”EXCPS”
dataType=”STRING”/>
                                <InputField name=”4” columnName=”DISKBLKS”
dataType=”STRING”/>
                                <InputField name=”5” columnName=”TAPEBLKS”
dataType=”STRING”/>
                                <InputField name=”6” columnName=”SRBTIME”
dataType=”STRING”/>
                                <InputField name=”7” columnName=”TCBTIME”
dataType=”STRING”/>
                                <InputField name=”8” columnName=”STCNAME”
dataType=”STRING”/>
                                <InputField name=”9” columnName=”PERIOD”
dataType=”STRING”/>
                                <InputField name=”10” columnName=”SYSID”
dataType=”STRING”/>
                                <InputField name=”11” columnName=”ACCOUNT”
dataType=”STRING”/>
                                <InputField name=”12” columnName=”DATE”
dataType=”DATETIME” format=”yyyyMMdd”/>
                            </InputFields>

                            <OutputFields>

                                <OutputField name=”Feed” src=”PARAMETER”
srcName=”Feed” />
                                <OutputField name=”headerstartdate” src=”INPUT”
srcName=”12” />
                                <OutputField name=”headerenddate” src=”INPUT”
srcName=”12” />
                                <OutputField name=”ACCOUNT” src=”INPUT” srcName=”11”
/>
                                <OutputField name=”PERIOD” src=”INPUT” srcName=”9” />
                                <OutputField name=”DATE” src=”INPUT” srcName=”12” />
                                <OutputField name=”STCNAME” src=”INPUT” srcName=”8”
/>
                                <OutputField name=”SYSID” src=”INPUT” srcName=”10” />
                                <OutputField name=”Z003” src=”INPUT” srcName=”1”
resource=”true” />
```

```
                                <OutputField name=”Z001” src=”INPUT” srcName=”2”
resource=”true” />
                                <OutputField name=”TSTEXCP” src=”INPUT” srcName=”3”
resource=”true” />
                                <OutputField name=”TSTSRBT” src=”INPUT” srcName=”6”
resource=”true” />
                                <OutputField name=”TSTTCBT” src=”INPUT” srcName=”7”
resource=”true” />
                                <OutputField name=”Z007” src=”INPUT” srcName=”5”
resource=”true” />
                                <OutputField name=”Z006” src=”INPUT” srcName=”4”
resource=”true” />
                            </OutputFields>

                            <Files>
                                <File name=”%ProcessFolder%/exceptionSTC.txt”
type=”exception” />
                            </Files>

                        </Input>
                        <Stage name=”CreateIdentifierFromIdentifiers” active=”true”
trace=”false” stopOnStageFailure=”true” >
                            <Identifiers>
                               <Identifier name=”Account_Code”>
                                   <FromIdentifiers>
                         <FromIdentifier name=”SYSID” offset=”1” length=”6”/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength=”true”/>
                               <Parameter modifyIfExists=”true”/>
                            </Parameters>
                        </Stage>
                        <Stage name=”CSRPlusOutput” active=”true”>
                            <Files>
                                <File
name=”%ProcessFolder%/server1/%LogDate_End%-TDSzSTC.txt”  />
                            </Files>
                        </Stage>
                    </Integrator>
                </Step>
                <Step id=”Integrator-TDS-RAFTSO” type=”ConvertToCSR”
programType=”java” programName=”integrator”
                    active=”true”>
```

```
                    <Integrator>
                        <Input name=”CollectorInput” active=”true”>

                            <Collector name =”TDS”>
                                <Connection dataSourceName=”DB8D”/>
                                            <Statement text=”SELECT
decimal(cpusec,20,6) as cpu, decimal(nsess) as numsess,
                                    decimal(excps) as excps,
                                    period, sysid, account, date
                                    FROM drl.raftso Where date &gt;= ? and date &lt;=
?”/>
                                <Parameter src=”PARAMETER” sqlType=”DATE”
position=”1” srcName=”StartLogDate”/>
                                <Parameter src=”PARAMETER” sqlType=”DATE”
position=”2” srcName=”EndLogDate”/>
                            </Collector>

                            <Parameters>
                                <Parameter name=”StartLogDate”
value=”%LogDate_Start%” dataType=”DATETIME” format=”yyyyMMdd”/>
                                <Parameter name=”EndLogDate” value=”%LogDate_End%”
dataType=”DATETIME” format=”yyyyMMdd”/>
                                <Parameter name=”Resourceheader” Value=”TDSzTSO”
dataType=”STRING”/>
                                <Parameter name=”Feed” value=”server1”
dataType=”STRING”/>
                                <Parameter name=”LogDate” value=”%LogDate_End%”
dataType=”DATETIME” format=”yyyyMMdd”/>
                            </Parameters>

                            <InputFields>
                                <InputField name=”1” columnName=”CPU”
dataType=”STRING”/>
                                <InputField name=”2” columnName=”NUMSESS”
dataType=”STRING”/>
                                <InputField name=”3” columnName=”EXCPS”
dataType=”STRING”/>
                                <InputField name=”4” columnName=”PERIOD”
dataType=”STRING”/>
                                <InputField name=”5” columnName=”SYSID”
dataType=”STRING”/>
                                <InputField name=”6” columnName=”ACCOUNT”
dataType=”STRING”/>
                                <InputField name=”7” columnName=”DATE”
dataType=”DATETIME” format=”yyyyMMdd”/>
```

```
                            </InputFields>

                            <OutputFields>

                                <OutputField name=”Feed” src=”PARAMETER”
srcName=”Feed” />
                                <OutputField name=”headerstartdate” src=”INPUT”
srcName=”7” />
                                <OutputField name=”headerenddate” src=”INPUT”
srcName=”7” />
                                <OutputField name=”ACCOUNT” src=”INPUT” srcName=”6”
/>
                                <OutputField name=”PERIOD” src=”INPUT” srcName=”4” />
                                <OutputField name=”SYSID” src=”INPUT” srcName=”5” />
                                <OutputField name=”TTSOSESS” src=”INPUT” srcName=”2”
resource=”true” />
                                <OutputField name=”TTSOEXCP” src=”INPUT” srcName=”3”
resource=”true” />
                                <OutputField name=”Z020” src=”INPUT” srcName=”1”
resource=”true” />
                            </OutputFields>

                            <Files>
                                <File name=”%ProcessFolder%/exceptionTSO.txt”
type=”exception” />
                            </Files>

                        </Input>
                        <Stage name=”CreateIdentifierFromIdentifiers” active=”true”
trace=”false” stopOnStageFailure=”true” >
                            <Identifiers>
                               <Identifier name=”Account_Code”>
                                   <FromIdentifiers>
                         <FromIdentifier name=”SYSID” offset=”1” length=”6”/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength=”true”/>
                               <Parameter modifyIfExists=”true”/>
                            </Parameters>
                        </Stage>
                        <Stage name=”CSRPlusOutput” active=”true”>
                            <Files>
```

```
                                <File
name="%ProcessFolder%/server1/%LogDate_End%-TDSzTSO.txt"  />
                            </Files>
                        </Stage>
                    </Integrator>
                </Step>
                <Step   id="Scan"
                                description="Scan TDSz"
                                type="Process"
                                programName="Scan"
                                programType="java"
                                active="true">
                               <Parameters>
                                <Parameter retainFileDate="false"/>
                                <Parameter allowMissingFiles="false"/>
                                <Parameter allowEmptyFiles="false"/>
                                <Parameter useStepFiles="false"/>
                               </Parameters>
                        </Step>
                        <Step   id="Process"
                                description="Standard Processing for TDSz"
                                type="Process"
                                programName="Bill"
                                programType="java"
                                active="true">
                    <Bill>
                       <Parameters>
                        <Parameter inputFile="CurrentCSR.txt"/>
                               </Parameters>
                            </Bill>
                        </Step>
                        <Step   id="DatabaseLoad"
                                description="Database Load for TDSz"
                                type="Process"
                                programName="DBLoad"
                                programType="java"
                                active="true">
                    <DBLoad>
                       <Parameters>
                               </Parameters>
                            </DBLoad>
                        </Step>
                        <Step   id="Cleanup"
                                description="Cleanup TDSz"
                                type="Process"
```

```
                                programName=”Cleanup”
                                programType=”net”
                                active=”false”>
                               <Parameters>
                                <Parameter DaysToRetainFiles=”45”/>
                                <Parameter cleanSubfolders=”true”/>
                               </Parameters>
                 </Step>
             </Steps>
         </Process>
     </Job>
</Jobs>
```

Sample job for z/OS user defined data load

```
<?xml version=”1.0” encoding=”utf-8”?>
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
    <Job    id=”Sample-TDSz-user-table”
            description=”Sample TDSz user table collection”
            active=”true”
            joblogWriteToDB=”false”
            joblogWriteToTextFile=”true”
            joblogWriteToXMLFile=”true”
            joblogShowStepOutput=”true”
            joblogShowStepParameters=”true”
            processPriorityClass=”Low”
            smtpServer=”mail.ITUAMCustomerCompany.com”
            smtpFrom=”ITUAM@ITUAMCustomerCompany.com”
            smtpTo=”John.ITUAMUser@ITUAMCustomerCompany.com”
            stopOnProcessFailure=”false”>
        <Process    id=”IBMTDSUSER”
                    description=”Process for TDSz Collection”
                    joblogShowStepOutput=”true”
                    joblogShowStepParameters=”true”
                    active=”true”>
            <Steps stopOnStepFailure=”true”>
                <Step id=”Integrator-TDS-USER” type=”ConvertToCSR” programType=”java”
programName=”integrator”
                    active=”true”>
                    <Integrator>
                        <Input name=”CollectorInput” active=”true”>
                            <Collector name =”TDS”>
                                <Connection dataSourceName=”DB8D”/>
```

```
                                <Statement text="SELECT DATE_END AS DATE,
                                                   MVS_SYSTEM_ID SYSID,
                                                   JOB_NAME AS JOBNAME,
                                                   ACCOUNT_FIELD1 AS ACCOUNT,
                                                 DECIMAL(SUM(ELAPSED_HOURS),20,6)
AS ELAPSEDH  ,
                                                 DECIMAL(SUM(CPU_SECONDS),20,6)
AS CPUSEC    ,
                                                 DECIMAL(SUM(ZAAP_SECONDS),20,6)
AS ZAAPSEC   ,
                                                 DECIMAL(SUM(ZIIP_SECONDS),20,6)
AS ZIIPSEC   ,
                                                 DECIMAL(SUM(IO_SERVICE_UNITS))
AS IOSU      ,
                                                 DECIMAL(SUM(JOB_COUNT))
AS JOBS
                                  FROM DRL.SAMPLE_ITUAM_V
                                         WHERE DATE_END BETWEEN ? AND ?
                                                 GROUP BY
                                                   DATE_END, MVS_SYSTEM_ID,
JOB_NAME, ACCOUNT_FIELD1"/>
                                <Parameter src="PARAMETER" sqlType="DATE"
position="1" srcName="StartLogDate"/>
                                <Parameter src="PARAMETER" sqlType="DATE"
position="2" srcName="EndLogDate"/>
                           </Collector>
                           <Parameters>
                               <Parameter name="StartLogDate"
value="%LogDate_Start%" dataType="DATETIME" format="yyyyMMdd"/>
                               <Parameter name="EndLogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                               <Parameter name="Resourceheader" Value="TDSzUSER"
dataType="STRING"/>
                               <Parameter name="Feed" value="TDSzUSER"
dataType="STRING"/>
                               <Parameter name="LogDate" value="%LogDate_End%"
dataType="DATETIME" format="yyyyMMdd"/>
                           </Parameters>
                           <InputFields>
                               <InputField name="1" columnName="DATE"
dataType="DATETIME" format="yyyyMMdd"/>
                               <InputField name="2" columnName="SYSID"
dataType="STRING"/>
                               <InputField name="3" columnName="JOBNAME"
dataType="STRING"/>
```

```
                        <InputField name=”4” columnName=”ACCOUNT”
dataType=”STRING”/>
                        <InputField name=”5” columnName=”ELAPSEDH”
dataType=”STRING”/>
                        <InputField name=”6” columnName=”CPUSEC”
dataType=”STRING”/>
                        <InputField name=”7” columnName=”ZAAPSEC”
dataType=”STRING”/>
                        <InputField name=”8” columnName=”ZIIPSEC”
dataType=”STRING”/>
                        <InputField name=”9” columnName=”IOSU”
dataType=”STRING”/>
                        <InputField name=”10” columnName=”JOBS”
dataType=”STRING”/>
                    </InputFields>
                    <OutputFields>
                        <OutputField name=”Feed” src=”PARAMETER”
srcName=”Feed” />
                        <OutputField name=”headerstartdate” src=”INPUT”
srcName=”1” />
                        <OutputField name=”headerenddate” src=”INPUT”
srcName=”1” />
                        <OutputField name=”ACCOUNT” src=”INPUT” srcName=”4”
/>
                        <OutputField name=”DATE” src=”INPUT” srcName=”1” />
                        <OutputField name=”SYSID” src=”INPUT” srcName=”2” />
                        <OutputField name=”JOBNAME” src=”INPUT” srcName=”3”
/>
                        <OutputField name=”ELAPSEDH” src=”INPUT” srcName=”5”
resource=”true” />
                        <OutputField name=”CPUSEC” src=”INPUT” srcName=”6”
resource=”true” />
                        <OutputField name=”ZAAPSEC” src=”INPUT” srcName=”7”
resource=”true” />
                        <OutputField name=”ZIIPSEC” src=”INPUT” srcName=”8”
resource=”true” />
                        <OutputField name=”IOSU” src=”INPUT” srcName=”9”
resource=”true” />
                        <OutputField name=”JOBS” src=”INPUT” srcName=”10”
resource=”true” />
                    </OutputFields>
                    <Files>
                        <File name=”%ProcessFolder%/exceptionTDSzUSER.txt”
type=”exception” />
                    </Files>
```

```
                        </Input>
                        <Stage name="CreateIdentifierFromIdentifiers" active="true"
trace="false" stopOnStageFailure="true" >
                            <Identifiers>
                               <Identifier name="Account_Code">
                                   <FromIdentifiers>
                         <FromIdentifier name="ACCOUNT" offset="1" length="6"/>
                                   </FromIdentifiers>
                               </Identifier>
                            </Identifiers>
                            <Parameters>
                               <Parameter keepLength="true"/>
                               <Parameter modifyIfExists="true"/>
                            </Parameters>
                        </Stage>
                        <Stage name="CSRPlusOutput" active="true">
                            <Files>
                                <File
name="%ProcessFolder%/TDSzUSER/%LogDate_End%-TDSzUSER.txt"/>
                            </Files>
                        </Stage>
                    </Integrator>
                </Step>
                <Step   id="Scan"
                                description="Scan TDSz"
                                type="Process"
                                programName="Scan"
                                programType="java"
                                active="true">
                               <Parameters>
                                <Parameter retainFileDate="false"/>
                                <Parameter allowMissingFiles="false"/>
                                <Parameter allowEmptyFiles="false"/>
                                <Parameter useStepFiles="false"/>
                               </Parameters>
                        </Step>
                        <Step   id="Process"
                                description="User Defined Processing for TDSz"
                                type="Process"
                                programName="Bill"
                                programType="java"
                                active="true">
                    <Bill>
                       <Parameters>
                        <Parameter inputFile="CurrentCSR.txt"/>
```

```
                            </Parameters>
                         </Bill>
                     </Step>
                     <Step   id=”DatabaseLoad”
                             description=”Database Load for TDSz”
                             type=”Process”
                             programName=”DBLoad”
                             programType=”java”
                             active=”false”>
                 <DBLoad>
                    <Parameters>
                            </Parameters>
                         </DBLoad>
                     </Step>
                     <Step   id=”Cleanup”
                             description=”Cleanup TDSz”
                             type=”Process”
                             programName=”Cleanup”
                             programType=”net”
                             active=”false”>
                            <Parameters>
                             <Parameter DaysToRetainFiles=”45”/>
                             <Parameter cleanSubfolders=”true”/>
                            </Parameters>
             </Step>
         </Steps>
     </Process>
 </Job>
</Jobs>
```

Sample job for AIX CPU burst calculation

```
<?xml version=”1.0” encoding=”utf-8”?>
<!--
 ***************************************************************
* AIXAA burst calculation for Redbook on IBM TUAM 7.1 by JSiglen 2007
 ***************************************************************
-->
<Jobs xmlns=”http://www.ibm.com/TUAMJobs.xsd”>
 <Job id=”AIXAA_BURST”
      description=”AIXAA aggregation and burst calculation”
      active=”true”
      joblogWriteToDB=”true” joblogWriteToTextFile=”true”
```

```
    joblogWriteToXMLFile="true" joblogShowStepOutput="true"
    joblogShowStepParameters="true" processPriorityClass="Low"
    stopOnProcessFailure="false">
<Process id="AIXAA_BURST05"
         description="Process for AIXAA data collection"
         joblogShowStepOutput="true"
         joblogShowStepParameters="true" active="true">
 <Steps stopOnStepFailure="true">
  <!-- read process files aacct1 -->
  <Step id="read_aacct1" type="ConvertToCSR"
        programName="integrator" programType="java" active="true">
   <Integrator>
    <Input name="AIXAAInput" active="true">
     <Files>
      <File name="%CollectorLogs%/lpar04/aacct1_%LogDate_End%.txt"/>
      <File name="/ti7b55/noback/ohm01/aacct1_%LogDate_End%.txt"/>
      <File name="/ti7b55/noback/ohm02/aacct1_%LogDate_End%.txt"/>
     </Files>
    </Input>
    <!-- pick the (date) hour info from header -->
    <Stage name="CreateIdentifierFromRegEx" active="true">
     <Identifiers>
      <Identifier name="DateHour">
       <FromIdentifiers>
        <FromIdentifier name="headerenddate"
                        regEx=".{4}-(.{2})-(.{2}).*" value="$1"/>
        <FromIdentifier name="headerenddate"
                        regEx=".{4}-(.{2})-(.{2}).*" value="$2"/>
        <FromIdentifier name="headerendtime"
                        regEx="(.{2}):.*" value="$1"/>
       </FromIdentifiers>
      </Identifier>
     </Identifiers>
     <Parameters>
      <Parameter modifyIfExists="true"/>
     </Parameters>
    </Stage>
    <!-- aggregate the data -->
    <Stage name="Aggregator" active="true">
     <Identifiers>
      <Identifier name="SysId"/>
      <Identifier name="DateHour"/>
      <Identifier name="SYSTEM_ID"/>
      <Identifier name="Partition_Name"/>
      <Identifier name="UserName"/>
```

```
     <Identifier name="Group"/>
    </Identifiers>
    <Resources>
     <Resource name="AAID0104"/>  <!-- CPU used -->
    </Resources>
    <Parameters>
     <Parameter defaultAggregation="false"/>
    </Parameters>
   </Stage>
   <Stage name="Sort" active="true">
    <Identifiers>
     <Identifier name="SYSTEM_ID" length="20"/>
     <Identifier name="DateHour" length="2"/>
    </Identifiers>
   </Stage>

   <Stage name="CSROutput" active="true">
    <Files>
     <File name="%ProcessFolder%/CurrentCSR_aacct1.txt"/>
    </Files>
   </Stage>
  </Integrator>
 </Step>

 <!-- read other aacct files -->
 <Step id="read aacctx" type="ConvertToCSR"
       programName="integrator" programType="java" active="true">
  <Integrator>
   <Input name="AIXAAInput" active="true">
    <Files>
     <File name="%CollectorLogs%/lpar04/aacct4_%LogDate_End%.txt"/>
     <File name="/ti7b55/noback/ohm01/aacct4_%LogDate_End%.txt"/>
     <File name="/ti7b55/noback/ohm02/aacct4_%LogDate_End%.txt"/>
    </Files>
   </Input>
   <!-- add record counter for average calculation -->
   <Stage name="CreateResourceFromValue" active="true">
    <Resources>
     <Resource name="BOOKED" value="100"/>
     <Resource name="MEMORY" value="512"/>
     <Resource name="RecordCount" value="1"/>
    </Resources>
    <Parameters>
     <Parameter modifyIfExists="true"/>
    </Parameters>
```

```
</Stage>
<!-- pick the (date) hour info from header -->
<Stage name=”CreateIdentifierFromRegEx” active=”true”>
 <Identifiers>
  <Identifier name=”DateHour”>
   <FromIdentifiers>
    <FromIdentifier name=”headerenddate”
                    regEx=”.{4}-(.{2})-(.{2}).*” value=”$1”/>
    <FromIdentifier name=”headerenddate”
                    regEx=”.{4}-(.{2})-(.{2}).*” value=”$2”/>
    <FromIdentifier name=”Hour”
                    regEx=”(.{2}).*” value=”$1”/>
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter modifyIfExists=”true”/>
 </Parameters>
</Stage>
<!-- aggregate the data -->
<Stage name=”Aggregator” active=”true”>
 <Identifiers>
  <Identifier name=”SysId”/>
  <Identifier name=”DateHour”/>
  <Identifier name=”SYSTEM_ID”/>
  <Identifier name=”Partition_Name”/>
 </Identifiers>
 <Resources>
  <Resource name=”AAID0402”/>   <!-- Entitelment -->
  <Resource name=”AAID0407”/>   <!-- Memory -->
  <Resource name=”BOOKED”/>
  <Resource name=”MEMORY”/>
  <Resource name=”RecordCount”/>
 </Resources>
 <Parameters>
  <Parameter defaultAggregation=”false”/>
 </Parameters>
</Stage>
<!-- calculate the average per memory-->
<Stage name=”ResourceConversion” active=”true”>
 <Resources>
  <Resource name=”MEMORY”>
   <FromResources>
    <FromResource name=”AAID0407” symbol=”a”/>
    <FromResource name=”RecordCount” symbol=”b”/>
```

```
      </FromResources>
     </Resource>
    </Resources>
    <Parameters>
     <Parameter formula="a/b"/>
    </Parameters>
   </Stage>
   <!-- calculate the booked CPU seconds -->
   <Stage name="ResourceConversion" active="true">
    <Resources>
     <Resource name="BOOKED">
      <FromResources>
       <FromResource name="AAID0402" symbol="a"/>
       <FromResource name="RecordCount" symbol="b"/>
      </FromResources>
     </Resource>
    </Resources>
    <Parameters>
     <Parameter formula="a/b*3600/100"/>
    </Parameters>
   </Stage>
   <Stage name="Sort" active="true">
    <Identifiers>
     <Identifier name="SYSTEM_ID" length="20"/>
     <Identifier name="DateHour" length="2"/>
    </Identifiers>
   </Stage>
   <Stage name="CSROutput" active="true">
    <Files>
     <File name="%ProcessFolder%/CurrentCSR_aacct.txt"/>
    </Files>
   </Stage>
  </Integrator>
 </Step>

 <Step id="generate CSR for BURST calculation" type="ConvertToCSR"
       programName="integrator" programType="java" active="true">
  <Integrator>
   <Input name="CSRInput" active="true">
    <Files>
     <File name="%ProcessFolder%/CurrentCSR_aacct1.txt"/>
     <File name="%ProcessFolder%/CurrentCSR_aacct.txt"/>
    </Files>
   </Input>
   <!-- aggregate the data -->
```

```
   <Stage name=”Aggregator” active=”true”>
    <Identifiers>
     <Identifier name=”SysId”/>
     <Identifier name=”DateHour”/>
     <Identifier name=”SYSTEM_ID”/>
     <Identifier name=”Partition_Name”/>
    </Identifiers>
    <Resources>
     <Resource name=”AAID0104”/>   <!-- CPU used -->
     <Resource name=”AAID0402”/>   <!-- Entitelment -->
     <Resource name="AAID0407"/>   <!-- Memory -->
     <Resource name=”BOOKED”/>
     <Resource name=”MEMORY”/>
    </Resources>
    <Parameters>
     <Parameter defaultAggregation=”false”/>
    </Parameters>
   </Stage>

   <!-- calculate CPU usage diff for splitting data later on -->
   <Stage name=”CreateResourceFromConversion” active=”true”>
    <Resources>
     <Resource name=”BURST”>
      <FromResources>
       <FromResource name=”AAID0104” symbol=”a”/>
       <FromResource name=”BOOKED” symbol=”b”/>
      </FromResources>
     </Resource>
    </Resources>
    <Parameters>
     <Parameter formula=”a-b”/>
     <Parameter modifyIfExists=”true”/>
    </Parameters>
   </Stage>
   <Stage name=”CSROutput” active=”true”>
    <Files>
     <File name=”%ProcessFolder%/CurrentCSR_for_Splitt.txt”/>
    </Files>
   </Stage>
  </Integrator>
 </Step>

 <!-- handling BURST records - include BURST great than 0 -->
 <Step id=”BURST calculation” type=”ConvertToCSR”
       programName=”integrator” programType=”java” active=”true”>
```

```
<Integrator>
 <Input name="CSRInput" active="true">
  <Files>
   <File name="%ProcessFolder%/CurrentCSR_for_Splitt.txt"/>
   <!-- dummy for each situation to avoid empty files -->
   <File name="%ProcessFolder%/DummyCSR.txt"/>
  </Files>
 </Input>
 <Stage name="IncludeRecsByValue" active="true">
  <Resources>
   <Resource name="BURST" cond="GT" value="0"/>
  </Resources>
 </Stage>
 <!-- do not handle recodrds with no entitelment -->
 <!-- there BURST will be >0 but is wrong -->
 <Stage name="ExcludeRecsByPresence" active="true">
  <Resources>
   <Resource name="AAID0402" exists="false"/>
  </Resources>
 </Stage>
 <Stage name="CreateResourceFromConversion" active="true">
  <Resources>
   <Resource name="beBOOKED">   <!-- below booked -->
    <FromResources>
     <FromResource name="BOOKED" symbol="b"/>
    </FromResources>
   </Resource>
  </Resources>
  <Parameters>
   <Parameter formula="b"/>
   <Parameter modifyIfExists="true"/>
  </Parameters>
 </Stage>
<Stage name="CSROutput" active="true">
 <Files>
   <File name="%ProcessFolder%/BURST/%LogDate_End%.txt"/>
  </Files>
 </Stage>
</Integrator>
</Step>
<!-- handling BURST=0 or non AAID0402 -->
<Step id="non BURST calculation" type="ConvertToCSR"
      programName="integrator" programType="java" active="true">
 <Integrator>
  <Input name="CSRInput" active="true">
```

```
   <Files>
    <File name=”%ProcessFolder%/CurrentCSR_for_Splitt.txt”/>
    <!-- dummy for each situation to avoid empty files -->
    <File name=”%ProcessFolder%/DummyCSR.txt”/>
   </Files>
  </Input>
  <Stage name=”IncludeRecsByValue” active=”true”>
   <Resources>
    <Resource name=”BURST” cond=”GE” value=”0”/>
   </Resources>
  </Stage>
  <Stage name=”ExcludeRecsByPresence” active=”true”>
   <Resources>
    <Resource name=”AAID0402” exists=”true”/>
   </Resources>
  </Stage>
  <Stage name=”CreateResourceFromConversion” active=”true”>
   <Resources>
    <Resource name=”beBOOKED”>   <!-- below booked -->
     <FromResources>
      <FromResource name=”BOOKED” symbol=”a”/>
     </FromResources>
    </Resource>
   </Resources>
   <Parameters>
    <Parameter formula=”a”/>
    <Parameter modifyIfExists=”true”/>
   </Parameters>
  </Stage>
  <Stage name=”DropFields” active=”true”>
   <Fields>
    <Field name=”BURST”/>   <!-- drop wrong BURST value -->
   </Fields>
  </Stage>

  <Stage name=”CSROutput” active=”true”>
   <Files>
    <File name=”%ProcessFolder%/BURSTno/%LogDate_End%.txt”/>
   </Files>
  </Stage>
 </Integrator>
</Step>

<!-- handling negative BURST - exclude BURST greater than 0 -->
<Step id=”negative BURST calculation” type=”ConvertToCSR”
```

```
      programName=”integrator” programType=”java” active=”true”>
 <Integrator>
  <Input name=”CSRInput” active=”true”>
   <Files>
    <File name=”%ProcessFolder%/CurrentCSR_for_Splitt.txt”/>
    <!-- dummy for each situation to avoid empty files -->
    <File name=”%ProcessFolder%/DummyCSR.txt”/>
   </Files>
  </Input>
  <Stage name=”ExcludeRecsByValue” active=”true”>
   <Resources>
    <Resource name=”BURST” cond=”GE” value=”0”/>
   </Resources>
  </Stage>
  <Stage name=”CreateResourceFromConversion” active=”true”>
   <Resources>
    <Resource name=”beBOOKED”>   <!-- below booked -->
     <FromResources>
      <FromResource name=”AAID0104” symbol=”a”/>
     </FromResources>
    </Resource>
   </Resources>
   <Parameters>
    <Parameter formula=”a”/>
    <Parameter modifyIfExists=”true”/>
   </Parameters>
  </Stage>
  <Stage name=”DropFields” active=”true”>
   <Fields>
    <Field name=”BURST”/>   <!-- drop negative BURST vlaue -->
   </Fields>
  </Stage>
  <Stage name=”CSROutput” active=”true”>
   <Files>
    <File name=”%ProcessFolder%/BURSTneg/%LogDate_End%.txt”/>
   </Files>
  </Stage>
 </Integrator>
</Step>

<!-- megre the files into one -->
<Step id=”Scan” description=”Scan AIXAA” type=”Process”
      programName=”Scan” programType=”java” active=”true”>
 <Parameters>
  <Parameter retainFileDate=”false”/>
```

```
   <Parameter allowMissingFiles="false"/>
   <Parameter allowEmptyFiles="false"/>
   <Parameter useStepFiles="false"/>
  </Parameters>
 </Step>

 <!-- finally do accoutning on the data -->
 <Step id="account conversion" type="ConvertToCSR"
       programName="integrator" programType="java" active="true">
  <Integrator>
   <Input name="CSRInput" active="true">
    <Files>
     <File name="%ProcessFolder%/CurrentCSR.txt"/>
    </Files>
   </Input>
   <!--  remove dummy entires -->
   <Stage name="ExcludeRecsByValue" active="true">
    <Identifiers>
     <Identifier name="SYSTEM_ID" cond="EQ" value="dummy"/>
    </Identifiers>
   </Stage>
   <!-- get account code from table based on SYSTEM_ID(hostname) -->
   <Stage name="CreateIdentifierFromTable" active="true">
    <Identifiers>
     <Identifier name="Account_Code_TMP">
      <FromIdentifiers>
       <FromIdentifier name="SYSTEM_ID" offset="1" length="10"/>
      </FromIdentifiers>
     </Identifier>
    </Identifiers>
    <Files>
     <File name="/opt/ibm/tuam/processes/Accttabl.txt"
           type="table"/>
     <File name="Exception_%LogDate_End%.txt"
           type="exception" format="CSROutput"/>
    </Files>
    <Parameters>
     <Parameter exceptionProcess="true"/>
     <Parameter sort="true"/>
     <Parameter upperCase="false"/>
     <Parameter writeNoMatch="false"/>
     <Parameter modifyIfExists="true"/>
    </Parameters>
   </Stage>
   <!--  add hostname as last part to the account code -->
```

```
<Stage name="CreateIdentifierFromIdentifiers" active="true">
 <Identifiers>
  <Identifier name="Account_Code">
   <FromIdentifiers>
    <FromIdentifier name="Account_Code_TMP
                    offset="1" length="40"/>
    <FromIdentifier name="SYSTEM_ID" offset="1" length="20"/>
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter modifyIfExists="true"/>
  <Parameter keepLength="true"/>
 </Parameters>
</Stage>
<!--  move the 9 character serial number to the SYSTEM_ID -->
<Stage name="CreateIdentifierFromRegEx" active="true" >
 <Identifiers>
  <Identifier name="SYSTEM_ID">
   <FromIdentifiers>
    <FromIdentifier name="SysId"
                    regEx="IBM,(.{9}).*" value="$1"/>
   </FromIdentifiers>
  </Identifier>
 </Identifiers>
 <Parameters>
  <Parameter modifyIfExists="true"/>
 </Parameters>
</Stage>
<Stage name="DropFields" active="false">
 <Fields>
  <Field name="Account_Code_TMP"/>
  <Field name="SysId"/>
  <Field name="DateHour"/>
 </Fields>
</Stage>
<Stage name="Sort" active="true">
 <Identifiers>
  <Identifier name="Account_Code" length="40"/>
  <Identifier name="SYSTEM_ID" length="20"/>
  <Identifier name="DateHour" length="2"/>
 </Identifiers>
</Stage>
<Stage name="CSRPlusOutput" active="true">
 <Files>
```

```
        <File name=”%ProcessFolder%/AcctCSR.txt”/>
       </Files>
      </Stage>
     </Integrator>
    </Step>
    <Step id=”Process”
          description=”Standard Processing for AIXAA” type=”Process”
          programName=”Bill” programType=”java” active=”true”>
     <Bill>
      <Parameters>
       <Parameter controlCard=”NORMALIZE CPU VALUES”/>
      </Parameters>
     </Bill>
    </Step>
    <Step id=”DatabaseLoad”
          description=”Database Load for AIXAA” type=”Process”
          programName=”DBLoad” programType=”java” active=”true”>
     <DBLoad>
      <Parameters>
      </Parameters>
     </DBLoad>
    </Step>
    <Step id=”Cleanup” description=”Cleanup AIXAA” type=”Process”
          programName=”Cleanup” programType=”java” active=”true”>
     <Parameters>
      <Parameter DaysToRetainFiles=”45”/>
      <Parameter cleanSubfolders=”true”/>
     </Parameters>
    </Step>
   </Steps>
  </Process>
 </Job>
</Jobs>
```

Example: A-1 DummyCSR.txt file for AIXAA burst calculation to avoid warnings

```
AATRID1,20071110,20071110,22:59:02,23:59:00,1,4,SYSTEM_ID,"dummy",SysId
,"IBM,dummy",Partition_Name,"dummy",DateHour,"111023",2,AAID0104,11.4,B
URST,11.4
AATRID1,20071111,20071111,08:59:02,09:59:00,1,4,SYSTEM_ID,"dummy",SysId
,"IBM,dummy",Partition_Name,"dummy",DateHour,"111109",6,AAID0104,4.7,AA
ID0402,360,AAID0407,24576,BOOKED,1080,MEMORY,2048,BURST,-1075.3
AATRID1,20071110,20071110,21:59:02,22:59:00,1,4,SYSTEM_ID,"dummy",SysId
,"IBM,dummy",Partition_Name,"dummy",DateHour,"111022",6,AAID0104,2090.7
,AAID0402,360,AAID0407,24576,BOOKED,1080,MEMORY,2048,BURST,1010.7
```

Sample script tuamdbsize.sh

```
#!/bin/bash
#**********************************************************************
# This script was written during the IBM TUAM Residency at ITSO in Austin
# October 1 - November 9 2007.
#**********************************************************************

#**********************************************************************
# Process the arguments.
#**********************************************************************
args="[ -d database ] [ -p prefix ] [ -t tempdir ] [ -o outdir ] [ -s spacereport ] [
-l loadreport ]"

database=ITUAMDB;
tabpfx=ITUAM;
ltrtable="CIMSLOADTRACKING";
tdir="/tmp";
odir="/opt/ibm/tuam/processes/ITUAMDB";
dt=`date +%Y%m%d`
spacerepf="$dt-spreport";
loadrepf="$dt-ldreport";

while getopts d:p:t:s:l:o: opt; do
        case $opt in
        d)  dbname=$OPTARG;;
        p)  tabpfx=$OPTARG;;
        t)  tdir=$OPTARG;;
        o)  odir=$OPTARG;;
        s)  spacerepf=$OPTARG;;
        l)  loadrepf=$OPTARG;;
        * ) echo "Parm error";
                exit 1;;
        esac
done

echo "Database $dbname";
echo "Prefix   $tabpfx";
echo "Tempdir  $tdir";
echo "Outdir   $odir";
echo "Spacerep $spacerepf";
```

```
echo "Loadrep  $loadrepf";

db2 connect to $dbname;
rc=$?
if [ $rc -ne 0 ]; then
    echo "Connect to $dbname failed rc $rc";
    exit 1
fi

#**********************************************************************
# Get a list of ITUAM tables
#**********************************************************************
sel1="select '$tabpfx' || '.' || name  from sysibm.systables where type='T' \
      and creator='$tabpfx' order by name";
db2 $sel1 | grep $tabpfx > $tdir/ituamtables.txt;

#**********************************************************************
# Get lists of row count, fixed columns lengths, variable columns lengths
#**********************************************************************
rm $tdir/ituamrows.txt
rm $tdir/ituamfcolsl.txt
rm $tdir/ituamvcolsl.txt
printf "Processing."
for line in `cat $tdir/ituamtables.txt`
do
   tab=`echo $line | tr  '.' ' ' | awk '{print $2}'`
   sel2="select 'ROWS', COUNT(*), '$tab' from $line"
   db2 $sel2 | grep ROWS >> $tdir/ituamrows.txt

   sel3="select 'FCOLS', SUM(LENGTH), '$tab' from sysibm.syscolumns where
TBNAME='$tab' \
         and TBCREATOR='$tabpfx' and COLTYPE not like 'VAR%'"
   db2 $sel3 | grep FCOLS >> $tdir/ituamfcolsl.txt

   sel4="select 'VCOLSN' || '.' || name from sysibm.syscolumns where TBNAME='$tab' \
         and TBCREATOR='$tabpfx' and COLTYPE like 'VAR%'"
   db2 $sel4 | grep VCOLSN > $tdir/ituamvcolsn.txt

   sel5="select 'VCOLS', VALUE(AVG(0)"
   for col in `cat $tdir/ituamvcolsn.txt`
   do
     vcol=`echo $col | tr '.' ' ' | awk '{print $2}'`
     sel5="$sel5 + AVG(LENGTH($vcol)+4)"
   done
   sel5="$sel5 ,0), '$tab' from $line"
```

```
   db2 $sel5 | grep VCOLS >> $tdir/ituamvcolsl.txt
   printf "."
done

#*********************************************************************
# Create database space report & csv file
#*********************************************************************
join -1 3 -2 3 $tdir/ituamrows.txt $tdir/ituamfcolsl.txt > $tdir/ituamrofc.txt
join -1 1 -2 3 $tdir/ituamrofc.txt $tdir/ituamvcolsl.txt | sort -k 3 -n -r >
$tdir/ituamrowscols.txt
printf "."

dt=`date`
printf "" > $odir/$spacerepf.csv
printf "$dt\n" > $odir/$spacerepf.txt
printf '%-30s%14s%14s%9s\n' "T a b l e    n a m e " "R o w s" "S i z e KB"
"Rows/MB">> $odir/$spacerepf.txt

tottbs=0
totrow=0
exec < $tdir/ituamrowscols.txt
while read line
  do
    tab=`echo $line | awk '{print $1}'`
    row=`echo $line | awk '{print $3}'`
    fcl=`echo $line | awk '{print $5}'`
    vcl=`echo $line | awk '{print $7}'`
    let "tcl = fcl + vcl"
    let "tbs = (tcl * row + 999) / 1000"
    rpmb=""
    if [ $row -ne 0 ]; then
       let "rpmb = 1000000 / $tcl"
    fi
    let "tottbs = tottbs + tbs"
    let "totrow = totrow + row"
    printf '%-30s%14d%14d%9s\n' $tab $row $tbs $rpmb >> $odir/$spacerepf.txt
    printf "$tab,$row,$tbs\n" >> $odir/$spacerepf.csv
  done
let "rpmb = 1000 * $totrow / $tottbs"
printf '%-30s%14d%14d%9s\n' "T o t a l" $totrow $tottbs $rpmb >> $odir/$spacerepf.txt

#*********************************************************************
# Create loadtracking report & csv file
#*********************************************************************
```

```
sel6="select 'LOADT', sourcefeed, filetype, sum(totalrecsloaded) from
"$tabpfx"."$ltrtable" \
      where datedeleted is NULL group by sourcefeed, filetype"
db2 $sel6 | grep LOADT > $tdir/ituamloadt.txt
printf "."

printf "" > $odir/$loadrepf.csv
printf "$dt\n" > $odir/$loadrepf.txt
printf '%-24s%-18s%14s\n' "S o u r c e f e e d" "F i l e t y p e" "R e c o r d s" >>
$odir/$loadrepf.txt

totrecs=0
exec < $tdir/ituamloadt.txt
while read line
  do
    feed=`echo $line | awk '{print $2}'`
    type=`echo $line | awk '{print $3}'`
    recs=`echo $line | awk '{print $4}'`
    let "totrecs = totrecs + recs"
    printf '%-24s%-18s%14d\n' $feed $type $recs >> $odir/$loadrepf.txt
    printf "$feed,$type,$recs\n" >> $odir/$loadrepf.csv
  done
printf '%-24s%-18s%14d\n' "T o t a l" " " $totrecs >> $odir/$loadrepf.txt

#**********************************************************************

printf "Done\n"
db2 disconnect $dbname;
rc=$?
if [ $rc -ne 0 ]; then
    echo "Disconnect from $dbname failed rc $rc";
    exit 1
fi

exit 0;
```

Sample script tuamdbstat.sh

```
#!/bin/bash
#**********************************************************************
# This script was written during the IBM TUAM Residency at ITSO in Austin
# October 1 - November 9 2007.
#**********************************************************************
```

```
#*******************************************************************
# Process the arguments.
#*******************************************************************
args="[ -t tempdir ] [ -o outdir ] [ -s statreport ] [ -i indir ]"

tdir="/tmp";
odir="/opt/ibm/tuam/processes/ITUAMDB";
idir="/opt/ibm/tuam/processes/ITUAMDB";
dt=`date +%Y%m%d`
statrepf="$dt-streport";

while getopts t:s:o:i: opt; do
        case $opt in
        t)  tdir=$OPTARG;;
        o)  odir=$OPTARG;;
        s)  statrepf=$OPTARG;;
        i)  idir=$OPTARG;;
        * ) echo "Parm error";
                exit 1;;
        esac
done

echo "Tempdir  $tdir";
echo "Outdir   $odir";
echo "Indir    $idir";
echo "Statrep  $statrepf";

#*******************************************************************
# Create space growth report
#*******************************************************************
prows=0
pssize=0

printf '%-20s%14s%14s%14s%14s\n' "T i m e s t a m p " "R o w s" "S i z e KB" "Growth
KB" "Rows/MB" > $odir/$statrepf.txt
for file in `ls -tr $idir/*spreport.txt`
do
   lh=`head -n 1 $file`
   lt=`tail -n 1 $file`
   rows=`echo $lt | awk '{print $6}'`
   size=`echo $lt | awk '{print $7}'`
   ts=`date -d "$lh" +%Y%m%d%H%M%S`
   if [ $prows -eq 0 ]; then
      printf '%-20s%14d%14d\n' "$ts" "$rows" "$size"  >> $odir/$statrepf.txt
```

```
   fi
   if [ $prows -ne 0 ]; then
      let "gsize = size - psize"
      let "rpmb = 1000 * $rows / $size"
      printf '%-20s%14d%14d%14d%14d\n' "$ts" "$rows" "$size" "$gsize" "$rpmb"  >>
$odir/$statrepf.txt
   fi
   prows=$rows
   psize=$size
done

exit 0;
```

Sample script tuamxa.bat

```
@echo off
rem *************************************************************** {COPYRIGHT-TOP}
rem * Licensed Materials - Property of IBM
rem * Restricted Materials of IBM
rem * IBM Tivoli Usage and Accounting Manager
rem * 5724-033, 5765-UAV, 5765-UA7, 44E7863
rem * (c) Copyright IBM Corp. 2004, 2007 All Rights Reserved.
rem *
rem * US Government Users Restricted Rights - Use, duplication or
rem * disclosure restricted by GSA ADP Schedule Contract with
rem * IBM Corp.
rem **************************************************************** {COPYRIGHT-END}
rem
rem ***************************************************************************
rem * Batch script to start JobRunner.
rem *
rem ***************************************************************************

setlocal
set TUAM_HOME=C:\IBM\tuam

call "%TUAM_HOME%\bin\setupEnv.bat"

set TUAM_INSTALLPATH=%TUAM_HOME%

rem ***** Passed in parameter ******
set JOB_FILE=%1
```

```
rem ***** TUAM Libraries **********
set JR_CLASSPATH=.;%CLASSPATH%
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucCommon.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucIntegrator.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucProcessEngine.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucJobFileConversion.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucDataAccess.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucJobRunner.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucVerification.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucRpd.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucExternalBridge.jar

rem ***** Message packages **********
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_ar.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_de.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_fr.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_it.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_es.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_ja.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_ko.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_pt_BR.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_zh_CN.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\aucMessages_zh_TW.jar

rem ***** External Libraries **********
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\auii18n.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\auibase.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\commons-net-1.4.1.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\icl.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\icu4j.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\jlanclient.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\mtftp.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\Notes.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\remoteaccess.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\ssh.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\jlog.jar

rem ***** ewas Linraries **************
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_HOME%\ewas\lib\j2ee.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_HOME%\ewas\lib\mail-impl.jar
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_HOME%\ewas\plugins\com.ibm.ws.runtime_6.1.0.jar
set
JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_HOME%\ewas\universalDriver\lib\db2jcc_license_cu.ja
r
```

```
rem ***** customer provided jar files *******
set JR_CLASSPATH=%JR_CLASSPATH%;%TUAM_LIB%\db2jcc.jar;%TUAM_LIB%\tuamxa.jar


"%JAVA_HOME%\jre\bin\java" -cp "%JR_CLASSPATH%" com.ibm.vbd.tuam.xa.TXAMethod %*

endlocal
```

TXAMethod Java class

```
package com.ibm.vbd.tuam.xa;

import java.util.StringTokenizer;
import java.io.File;

import com.ibm.tivoli.tuam.exceptions.AcctMgrInvalidParameterException;
import com.ibm.tivoli.tuam.jobrunner.*;

public class TXAMethod {
   private String tws_CPU, tws_Host, tws_Master;
   private String formattedSchedDate, universalDate;
   private String jobName, jobId;
   private String streamLogon, nodeName, stdList, portNumber;
   private String testOptions, scheduleName, taskName;
   private String currentRunNumber, requiredRunNumber;
   private String runArgs;
   private boolean debug = true;

   public int launch(String xmlFile) {
      int rc = 0;
      String jrArgs[] = { xmlFile };
      try {
         JobRunner jr = new JobRunner(jrArgs);
         rc = jr.run();
      } catch (AcctMgrInvalidParameterException e) {
         e.printStackTrace();
      }

      return rc;
   }
```

```
   public int execute() {
      //*
      // This perform the TWS extended agent function
      // CF = check file
      // MJ (manage job) is not implemented
      // CC (check connection) is always successful, useful for a remote system
connection
      // LJ = launch job, invoke the launch method
      //*
      int rc = 0;
      if (taskName.equals("CF")) {
         // Check file runArgs = filename
         File f = new File(runArgs);
         if (f.exists()) rc = 0;
         else rc = 1;
      } else if (taskName.equals("MJ")) {
         // Manage job runArgs = jobid|stdlist
         // not applicable
         rc = 1;
      } else if (taskName.equals("LJ")) {
         // Launch job runArgs = scriptname
         System.out.println("%CJ EXEC");
         rc = launch(runArgs);
      } else if (taskName.equals("CC")) {
         // Check connection
         // always successful
         rc = 0;
      } else {
         System.out.println("Unknown task parameter "+taskName);
         System.err.println("Unknown task parameter "+taskName);
         rc = 1;
      }
      if (rc == 0) {
         System.out.println("%JS");
      } else {
         System.out.println("%CJ ABEND");
      }
      return rc;
   }

   public TXAMethod(String[] args) {
      //*
      // initialize all TWS variables to an empty String
      // to avoid NullPointerException
      //*
```

```
      tws_CPU = new String(""); tws_Host = new String(""); tws_Master = new
String("");
      formattedSchedDate = new String(""); universalDate = new String("");
      jobName = new String(""); jobId = new String("");
      streamLogon = new String(""); nodeName = new String(""); stdList = new
String(""); portNumber = new String("");
      testOptions = new String(""); scheduleName = new String(""); taskName = new
String("");
      currentRunNumber = new String(""); requiredRunNumber = new String("");
      //*
      // runArgs will be the content of TaskName in the Job definition
      //*
      runArgs = new String("");
      //*
      // parse the arguments
      //*
      for (int i=0;i<args.length;i++) {
         if (args[i].equals("-c")) {
            i++;
            // TWS agent detail (not used)
            StringTokenizer st = new StringTokenizer(args[i],",");
            tws_CPU = st.nextToken();
            tws_Host = st.nextToken();
            tws_Master = st.nextToken();
         } else if (args[i].equals("-d")) {
            i++;
            // schedule date; may be used later for JobRunner date
            StringTokenizer st = new StringTokenizer(args[i],",");
            formattedSchedDate = st.nextToken();
            universalDate = st.nextToken();
         } else if (args[i].equals("-j")) {
            i++;
            // job name and id parameters (to determine rerun etc)
            StringTokenizer st = new StringTokenizer(args[i],",");
            jobName = st.nextToken();
            jobId = st.nextToken();
         } else if (args[i].equals("-l")) {
            i++;
            // logon user, may be compared with TUAM user
            streamLogon = args[i];
         } else if (args[i].equals("-n")) {
            i++;
            // TWS node name (not used)
            nodeName = args[i];
         } else if (args[i].equals("-o")) {
```

```
        i++;
        // output stdlist file name (not used)
        stdList = args[i];
     } else if (args[i].equals("-p")) {
        i++;
        // agent port number (not used)
        portNumber = args[i];
     } else if (args[i].equals("-q")) {
        i++;
        // get test Options (not used)
        testOptions = args[i];
     } else if (args[i].equals("-r")) {
        i++;
        // get Run numbers (not used)
        StringTokenizer st = new StringTokenizer(args[i],",");
        currentRunNumber = st.nextToken();
        requiredRunNumber = st.nextToken();
     } else if (args[i].equals("-s")) {
        i++;
        // get Job stream name (not used)
        scheduleName = args[i];
     } else if (args[i].equals("-t")) {
        i++;
        // define XA operation mode
        taskName = args[i];
     } else if (args[i].equals("-V")) {
        i++;
        // print version
        this.printBanner();
        System.exit(0);
     } else if (args[i].equals("--")) {
        // end of TWS arguments, collect TaskName string
        StringBuffer a = new StringBuffer();
        i++;
        if (i<args.length) {
           do {
              a.append(args[i]);
              i++;
           } while (i<args.length);
           runArgs = a.toString();
        }
        else runArgs = new String("");
     } else {
        // echo usage
        this.printError("Invalid argument found "+args[i]);
```

```
            System.out.println("%CJ FAIL");
            System.exit(1);
         }

         if (taskName == null || taskName.length()==0) {
            System.out.println("Command must be invoked using TWS method");
            System.out.println("Invalid task or -t not found");
            System.out.println("%CJ FAIL");
            System.exit(1);
         }

         // argument parsing done
      }
      if (debug) printDebugInfo();
   }

   public void printError(String errmsg) {
      //*
      // Output the error message
      //*
      System.out.println("Command must be invoked using TWS method");
      System.out.println(errmsg);
      System.out.println("Supported arguments: -cdjlnopqrstV");
      System.out.println("See SG24-6030");
   }

   public void printBanner() {
      //*
      // Banner
      //*
      System.out.println("IBM Tivoli Usage and Accounting Manager V7.1");
      System.out.println("Access method for IBM Tivoli Workload Scheduler");
      System.out.println("(c) IBM Technical Support Organization (ITSO)");
      System.out.println("(c) Budi Darmawan");
      System.out.println("(c) 2007");
   }

   public void printDebugInfo() {
      //*
      // Print arguments and other debugging information
      //*
      System.out.println("TWS Access Method for TUAM");
      System.out.println("TWS detail: "+ tws_CPU+","+ tws_Host+","+ tws_Master);
      System.out.println("Schedule date: "+ formattedSchedDate+","+ universalDate);
      System.out.println("Job:" + jobName+","+ jobId);
```

```
      System.out.println("Node information:"+ streamLogon+","+ nodeName+","+
stdList+","+ portNumber);
      System.out.println("Tasks: "+ testOptions+","+ scheduleName+","+ taskName);
      System.out.println("Run number: "+ currentRunNumber+","+ requiredRunNumber);
      System.out.println("Arguments: "+ runArgs);
   }

   public static void main(String[] args) {
      TXAMethod ta = new TXAMethod(args);
      System.exit(ta.execute());
   }

}
```

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247404`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247404.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG247404.zip	Zipped sample xml files and scripts as listed in Appendix A, “Sample listings and programs” on page 285

System requirements for downloading the Web material

The sample codes are for IBM Tivoli Usage and Accounting Manager. The system requirement correspond to what is required to run an IBM Tivoli Usage and Accounting Manager server. The following system additional requirement is recommended:

Hard disk space: 2MB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Abbreviations and acronyms

AIX	Advanced Interactive eXecutive
AIXAA	AIX Advanced Accounting
ARM	Application Response Measurement
BIRT	Business Intelligence and Reporting Tools
CMDB	Configuration Management Database
CPU	Central Processing Unit
CSR	Common Source Format
DB2	Database 2™
DDF	Distributed Data Facility
FTP	File Transfer Program
GB	Giga Bytes
GUI	Graphical User Interface
HFS	Hierarchical File System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure
I/O	Input Output
IBM	International Business Machines Corp.
IIS	Internet Information Server
IP	Internet Protocol
ISC	Integrated Solution Console
IT	Information Technology
ITIL	Information Technology Infrastructure Library
ITSO	International Technical Support Organization
JCL	Job Control Language
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
LPAR	Logical partition
LU	Logical Unit
MB	Mega Byte
MVS™	Multiple Virtual Storage
ODBC	Open Database Connectivity
OGC	Office of Government Commerce
OS	Operating System
POC	Proof of Concept
RAF	Resource Accounting Facility
ROI	Return on Investment
RPD	Remote Product Deployment
SDK	Software Development Kit
SFTP	Secure FTP
SMF	System Measurement Facility
SOA	Service Oriented Architecture
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
TDS	Tivoli Decision Support
TPC	TotalStorage Productivity Control
TSO	Time Sharing Option
TUAM	Tivoli Usage and Accounting Manager
TWS	Tivoli Workload Scheduler
UDB	Universal Database
URL	Universal Resource Locator
VCDB	Virtual Center database
VIOS	Virtual I/O Server
XML	eXtensible Markup Language

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see "How to get Redbooks" on page 374. Note that some of the documents referenced here may be available in softcopy only.

- *Deployment Guide Series: Tivoli Usage and Accounting Manager V7.1*, SG24-7569
- *Accounting and Chargeback with Tivoli Decision Support for OS/390*, SG24-6044

Other publications

These publications are also relevant as further information sources:

- *Tivoli Usage and Accounting Manager Quick Start Guide,* GC23-6188
- *Program Directory for Tivoli Decision Support for z/OS (English) V01.08.00,* GI11-4249
- *Tivoli Usage and Accounting Manager Data Collectors for UNIX and Linux User's Guide Version 6.1.1*, SC32-1556
- *Tivoli Usage and Accounting Manager Data Collectors for Microsoft Windows User's Guide Version 6.1.1*, SC32-1557
- *Installation and Upgrade Guide ESX 3.0.1 and VirtualCenter 2.0.1*, VMware Item: VI-ENG-Q306-292
- *Handbuch zur Server-Konfiguration ESX Server 3.0.1 und VirtualCenter 2.0.1*, Artikelnummer: VI-DEU-Q406-314
- *Server Configuration Guide ESX Server 3.0.1 and VirtualCenter 2.0.1*, VMware Item: VI-ENG-Q206-215
- *Virtual Infrastructure Web Access Administrator's Guide ESX Server 3.0.1 and VirtualCenter Server 2.0.1*, VMware Item: VI-ENG-Q306-294

Online resources

These Web sites are also relevant as further information sources:

- Tivoli Usage and Accounting Manager publication center

  `http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?topic=/com.ibm.ituam.doc_7.1/welcome.htm`

- Tivoli Usage and Accounting Manager Web site

  `http://www-306.ibm.com/software/tivoli/products/usage-accounting/`

- VMware SDK Reference Guide

  `http://pubs.vmware.com/vi301/sdk/ReferenceGuide/index.html`

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

W

X

Z

IBM Tivoli Usage and Accounting Manager V7.1 Handbook

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

IBM Tivoli Usage and Accounting Manager V7.1 Handbook

Financial management solution for IT related services

End-to-end IT charging and accounting solution

Works with Tivoli Decision Support for z/OS

Financial management of IT resources allows an IT department to be transformed from a cost center to a service provider. One aspect of this is usage accounting, which helps the IT department understand the usage patterns of its customers or users and allows for service charges that reflect that usage. In addition, usage data demonstrates how IT operations can be optimized to increasing efficiency.

Tivoli Usage and Accounting Manager provides the tools to perform data collection and accounting for IT-related usage from various sources. It even allows the custom integration of data from non-standard format sources. It supports the whole life cycle of financial management from budgeting to usage accounting and billing, to reporting.

This book will help you understand, install, configure, and use the new IBM Tivoli Usage and Accounting Manager V7.1.

The discussion starts with an overview of Tivoli Usage and Accounting Manager concepts and capabilities along with the structure of the product. The installation and verification of each component is presented in detail. Sample scenarios are executed and explained, including Operating System usage collection, virtual environment collection (VMware ESX server and System p partitioning), and Tivoli Decision Support for z/OS interface.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7404-00 ISBN 0738485640