Writing a Data Supplier for the Alert Monitor

Copyright

Contents

Introduction

With the new CCMS alert monitor and monitoring architecture in R/3 Release 4.0, you can easily add monitoring and alert functionality for any R/3 component or external hardware or software component.

Adding "monitoring and alert functionality" for a component means adding the component to the CCMS R/3 alert monitor. An R/3 system operator can then use the alert monitor (and other CCMS monitors) to watch over the condition of the component, display performance or "current condition" information on the component, and manage alerts (warnings and problems) pertaining to the component.

Anyone (SAP, partner, or customer developer) can add a component to the alert monitor by writing a data supplier program. A data supplier program plugs into the CCMS monitoring architecture by way of an ABAP or C interface. The data supplier then passes information on the component that is to be monitored to the monitoring architecture. The monitoring architecture, in turn, makes the component, its data, and its alerts visible in the alert monitor and in other "data consumers" of the monitoring architecture..

This document explains how to write a data supplier using the ABAP interface provided by the monitoring architecture

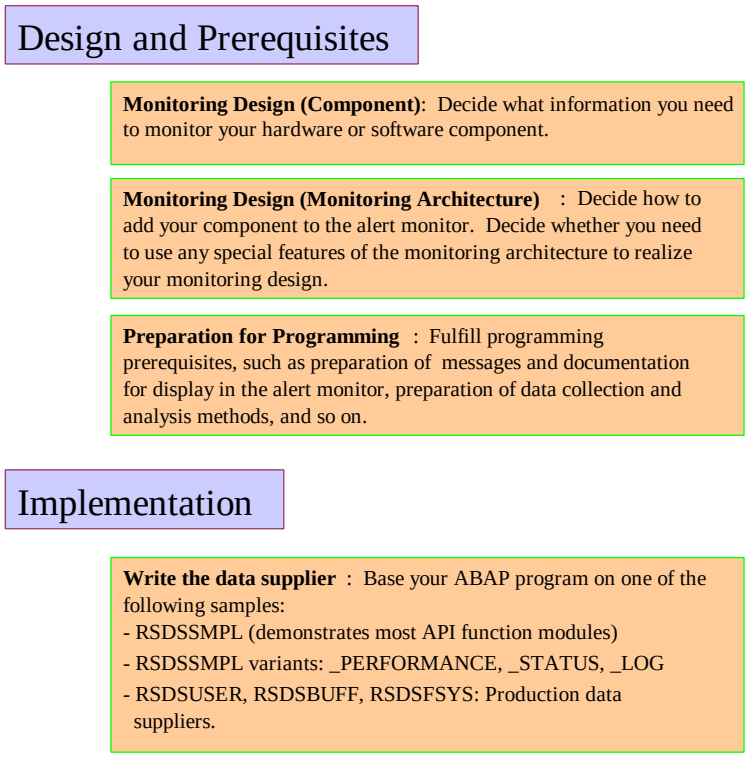

Figure 1. Writing a Data Supplier: Overview.

Terminology note: In the monitoring architecture, we use "**data supplier**" as the generic term for a program that adds monitoring functionality to and reports information into the alert monitor. We will use the term "data supplier" to refer to all programs that supply monitoring functionality.

We will, however, often distinguish between two types of data suppliers. These are as follows:

- **Passive data suppliers**, or **data collection methods**, are those data suppliers that are started by the monitoring architecture and which are formally defined in the monitoring architecture. "Passive" describes the behavior of the data supplier with respect to the monitoring architecture: the data supplier must be started by the monitoring architecture. We will use the term "data collection methods" to refer to these data suppliers.

- **Active data suppliers** are those data suppliers that are started by the monitored application, and not by the monitoring architecture. These data suppliers are "active" in their start behavior with respect to the monitoring architecture. We will use the term "active data supplier" for these programs.

Comments or more information: We welcome your comments, suggestions, and questions. Mail to us at ccms@sap-ag.de.

Changes in this Edition

- TOOL_TOOLDISPATCHER parameter in SALI CREATE_ATTACH calls is being removed as of Release 4.6A. Please change any existing function module calls accordingly. ABAP handles calls with superfluous parameters correctly, but TODO errors (internal SAP error checking) result.

  The parameter was never used, because the assignment of a method to a dispatcher is made in the method definition in transaction RZ21.

- Automatic numbering: The new recommended procedure for generating system-unique numbers for MTE TIDs is automatic permanent assignment (CREATE_ATTACH value AL_NR_AUTO) by the monitoring architecture. See *Number Range Management* 39.

- New MTEs:

  - Text attributes (display text without triggering alerts). See *Decision Table: Monitoring Attributes for Text Attributes* on page 27

  - References (include referenced MTEs in a monitoring tree that a data supplier sets up). See *Structuring Your Monitoring Tree: Summaries and References* on page 45.

  - Summary MTEs (headings for organizing a monitoring tree that a data supplier sets up). See *Structuring Your Monitoring Tree: Summaries and References* on page 45.

- SAP MAIL: As of Release 4.6A, it is possible to have an autoreaction method send mail by way of SAP Office. See *Auto-Reaction Methods* on page 58.

- Client-specific contexts and system-global contexts. See *Creating Your Own Monitoring Context* on page 56.

- Implementation techniques and tips:

  - Implementing methods as function modules. See *Preparing a Function Module for Use as an Alert Monitor Method* on page 58.

  - Auto-reaction methods (see above).

  - Self-Monitoring Messages (how to issue messages in case of problems in a method). See *Issuing Self-Monitoring Messages* on page 58.

- The method definition documentation has been revamped. See *Defining Your Methods in the Monitoring Architecture* on page 5.
- Deleting MTEs during development of a data supplier, in order to make changes in the permanent MTEs that the data supplier creates. See Discarding a Monitoring Object and Attributes on page 57.

Implementation Checklist

Selecting Monitoring Data & Methods

The monitoring architecture is conceived as an infrastructure. That means that monitoring functionality is decentralized. There is no central, monolithic monitoring program. Rather, monitoring functionality is written by the people who know the components the best: you, the component programmers.

You provide the monitoring data. You also provide the monitoring "methods" -- the data supplier, an analysis tool, and perhaps also an automatic reaction tool. The monitoring architecture provides the infrastructure, such as display and alert management services.

The point: the quality and usefulness of the monitoring data for your component depends entirely upon you. It is up to you to decide:

- What information a user needs to monitor and manage your component. Implicitly, you need to decide what it means to monitor and manage your component. Does it mean watching a performance value? Waiting for the appearance or non-appearance of certain messages? A combination of both types of information gathering?

  Information resources required:

  _____

  _____

  _____

- How to collect this information reliably and accurately. Do you have a function module or report that collects the information that you need? And can you build this function module or report into your data supplier? Asked another way, do you have the instrumentation in your component that is needed to gather the required data? Also, can you ensure under all reasonable circumstances that you can reliably supply accurate information?

  Available / required data collection instrumentation / technology:
  _____

- How to let the user respond to problems (alerts). Do you have an "analysis method" – a transaction, report, or external operating system command – with which the user can gather any additional information necessary and / or take any actions necessary to correct a problem?

  Analysis method: _____

- Do you have any tool that is capable of responding automatically to an alert (auto-reaction method)? Or is there a standard action that should be taken if an alert occurs (for example, notify administrator, send mail to a call center).

  Auto-reaction method / action:
  _____

Defining Your Methods in the Monitoring Architecture

Now that you know what methods you wish to use, here is how to make the methods known to the monitoring architecture. You will need to do the following:

- Define the method in the monitoring architecture, in the customizing transaction (RZ21). This step is necessary only if your method should be startable from the alert monitor.

  A method definition is needed if a method is to be started automatically by the monitoring architecture (active data supplier, automatic reaction method). A method definition is also needed if you want users to be able to start a method by hand from the monitoring architecture. In general: passive data collection methods, analysis methods, and auto-reaction methods must be defined as methods. Active data collection methods may be, but need not be, defined as methods.

  For more information, see *Defining Methods in Transaction RZ21* on page 37.

- Associate the method(s) with the monitoring functionality that you are adding to the monitoring architecture.

  You must do this for any method that is to be accessible from the monitoring architecture, whether it is to be automatically started by the monitoring architecture or manually started by users.

  You associate methods with your monitoring functionality directly in your data supplier program. You make these specifications in the data supplier's interface to the monitoring architecture. You can also make this association manually in transaction RZ21, if necessary.

  When you associate a data supplier with your monitoring functionality, you can also specify how the data supplier is to be dispatched by the monitoring architecture. These specifications need only be made if the data supplier is to function as a passive data collection method, a method that is started automatically by the monitoring architecture.

  For more information, see *Specifying Methods to Use in Your Data Supplier* on page 34.

Data Supplier Implementation Checklist

Note: Fast Path. No time to make design decisions right now? Or do you just want to prototype quickly? Then copy and alter to suit one of these sample reports:

- Rsdssmpl_performance: monitoring object and performance attribute
- Rsdssmpl_status: monitoring object and status (single-message) attribute
- Rsdssmpl_log: monitoring object and log (message container) attribute

Fill out this table to make the design decisions and to provide the information that you will need to implement your data supplier. Take the completed table and use it to adapt one of the sample reports available in the system and at the end of this paper. Your data supplier is then ready to use.

The table starts with organizational and structural decisions. Your decisions here flow into the "CREATE_ATTACH" function module calls for monitoring objects and attributes.

The table then continues with the object- and attribute-specific implementation decisions you will need to make. For example, you need to decide on alert thresholds and to assign criticalities to status and log messages.

Decision Table: Organizational Decisions

Design Consideration / Info Requirement	**Choices / Specifications**
Monitoring data: Which kinds of information do you want to report?	__ **Values for one or more performance parameters,** and / or __ **Single discrete messages** (such as individual status, warning, or error messages), and / or __ **Messages in context (logs or traces)**. If your component may issue multiple related messages, then you can use a log attribute instead of a status attribute to report messages to the alert monitor. The advantage: the log attribute presents the messages as a log. The user can see the messages in context. You can use a log attribute to implement a log or trace for your component. The monitoring architecture provides the log functionality. All that you have to do is report your messages into the log attribute. __ **Text attributes.** Infrequently changed textual information on a monitoring object (ID, for example). No alert functionality available. Your answers determine the function modules from the monitoring architecture that you will need to use. Implementation help: See *Type of Information to Provide: Performance, Status Messages, Log/Trace* on 31.

Design Consideration / Info Requirement	**Choices / Specifications**
Scope of your data supplier	Mark the applicable choice: __ Is your data supplier **system-wide** in scope? Is it to be started only once per R/3 System? __ Is your data supplier **R/3 instance-local or host-server local**? Is it to be started once per R/3 instance or once per host system? Implementation help: See *Scope of Your Data Supplier* on page 32.

Dispatching your data supplier How should your program be started?	**Should the alert monitor start the data supplier (passive or active data supplier)?**
	
	__ The alert monitor should run the collection method every ___ seconds in a dialog work process.
	
	__ The alert monitor should run the collection method every ___ seconds in a background work process.
	
	__ My component will run the data supplier as needed to report new values.
	
	Status change if the data supplier does not run?
	
	__ Yes. The status of the monitoring attribute and object should switch to "inactive" if the data supplier is not heard from within __ seconds.
	
	__ No. The status of the attribute should not be set to inactive. The data supplier must not necessarily supply information within a set time period.
	
	Sample Code:
	
	`CALL FUNCTION SALI_PERF_CREATE_ATTACH...`
	`  MTE_SECONDS_TIL_COLLECTINGTOOL = 240 " (sec.)`
	`  " Time period for restart, 0 for manual start`
	`  TOOL_TOOLDISPATCHER = 'SAP_CCMS_DEFAULT_TD'`
	`  " Default: Alert monitor starts DS in Dialog`
	`  " Or: 'SAP_CCMS_BATCH_DISPATCHER' or`
	`  " 'SAP_CCMS_ONLY_MANUALLY_DP'`
	`  MTE_SECONDS_UNTIL_SET_INACTIVE = 900`
	`  " Is element active?`
	`  " If no value is reported for 900 seconds,`
	`  " the MTE in the monitor goes white:`
	`  " Inactive. Set to 0 to switch off.`
	
	Decision help: See *Dispatching Your Data Supplier* on page 35.

Design Consideration / Info Requirement	Choices / Specifications
Dispatching II: These additional options affect how and where a collection method runs, but must be set in the tool definition with transaction RZ21.	**Autostart at system start?** __ Run the collection method automatically when R/3 starts. __ My component will run the data supplier as needed. **Where should the collection method run (choose one)?** __ On the server/instance specified in an MTE. __ On any server/instance. __ On the database server. __ At a particular RFC destination. See also *Scope of Your Data Supplier* on page 32. **If the collection method is to be started by the alert monitor, should it be started** __ Once for each individual MTE for which it is due to run? __ Once for all MTEs for which it is due to run? These options need to be set by hand in the definition of a collection method in the alert monitor (transaction RZ21). More help: See *Defining Methods in Transaction RZ21* on page 37.

Design Consideration / Info Requirement	**Choices / Specifications**

Number range management: A monitored component is identified by ID number in the alert monitor.
 	
Should you or should the monitoring architecture manage the numbers?
 	
Recommendation: Have the monitoring architecture manage number ranges (automatic permanent numbering). Self-managed permanent number ranges have been superseded by automatic numbering.
	Mark the applicable choice:
	
	__ **The alert monitor should manage the number range** for my monitoring objects (temporary objects, or automatic permanent numbering).
	
	__ **My component will manage the number range** (permanent number range – not recommended).
	
	Sample Code: For the monitoring object
	
	`CALL FUNCTION 'SALI_MO_CREATE_ATTACH'`
	`...`
	`  NUMRANGE = AL_NR_AUTO " permanent number range`
	`  " Sample uses the automatic number`
	`  " range for the MO handle. Benefit:`
	`  " the monitoring architecture`
	`  " manages the number range; you`
	`  " don't have to.`
	`  UNIQUENUM = 0`
	`  " ID number. Set to 0 if using`
	`  " numrange = al_nr_temp. Omit if using`
	`  " al_nr_auto.`
	
	For each monitoring attribute:
	
	`CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'`
	`...`
	`  MTE_NUMRANGE  = AL_NR_AUTO " See above.`
	`  MTE_UNIQUENUM = 0          " See above.`
	
	Implementation help: See Number Range Managementon page 39.

Design Consideration / Info Requirement	**Choices / Specifications**

Messages: For describing and for providing F1 help for your monitoring objects, monitoring attributes, and alerts	T100 Message ID __ and number ___ **for describing your monitoring object** (your component) in the alert monitor
	
	T100 Message ID __ and number ___ **for describing each monitoring attribute** (type of data reported) in the alert monitor
	
	T100 Message ID __ and number ___ **for explaining alerts** generated with respect to a particular monitoring attribute
	
	Sample code: For the monitoring object:
	
	`CALL FUNCTION 'SALI_MO_CREATE_ATTACH'`
	`...`
	`  DESC_TEXT_MSGID = 'RT'`
	`  DESC_TEXT_MSGNO = 092`
	`  " T100-Message: Display as descript.`
	`  " and F1-help anchor for MO (SE91)`
	
	For the monitoring attributes and alerts:
	
	`CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'`
	`...`
	`  MTE_F1_HELP_TEXT_MSGID = 'RT'`
	`  MTE_F1_HELP_TEXT_MSGNO = 093`
	`  " T100-Message to display`
	`  " as descript. and F1-help`
	`  " anchor for MA (SE91).`
	`  PERF_ALERT_TEXT_MSGID = 'RT'`
	`  PERF_ALERT_TEXT_MSGNO = 001`
	`  " T100-Message to display`
	`  " on alert and as F1-help`
	`  " anchor for alert (SE91).`
	
	Messages in status and log attributes are R/3 “T100 messages”. These messages therefore automatically have access to F1 help.
	
	Implementation help: See *Messages in the Alert Monitor* on page 41.

Design Consideration / Info Requirement	**Choices / Specifications**

Properties: MTE Classes and Customizing Groups	**MTE classes** let you share “general properties”, “method assignment” settings (transaction RZ21), and F1-documentation among instances of a monitoring object or monitoring attribute. The MTE class is also used to select MTEs with "rule-based" nodes in monitor definitions.
	
	Guideline: Each type of monitoring object or attribute should have its own MTE class.
	
	Name for the MTE class for the object:
	_____
	
	Names for the MTE classes for each of the monitoring attributes:
	
	_____
	_____
	
	Choose a **customizing group** for each monitoring attribute. The customizing group allows sharing of alert thresholds across instances of a monitoring attribute:
	
	_____
	_____
	
	Sample Code: For the monitoring object:
	
	`CALL FUNCTION 'SALI_MO_CREATE_ATTACH'`
	`...`
	`  MTE_CLASS = 'OBJECT_CLASS'`
	
	For each monitoring attribute
	
	`CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'`
	`...`
	`  MTE_CLASS = 'PERF_ATTRIBUTE_CLASS'`
	`  PERF_CUSTOMIZING_GROUP = 'MY_PERF_CUST_GRP'`
	`  " Name for alert threshold customizing for`
	`  " this attribute (cf. MTE Class, above).`
	
	`CALL FUNCTION 'SALI_SMES_CREATE_ATTACH'`
	`...`
	`  MTE_CLASS = 'SMES_ATTRIBUTE_CLASS'`
	`  SMES_CUSTOMIZING_GROUP = 'MY_SMES_GROUP'`
	
	`CALL FUNCTION 'SALI_MSC_CREATE_ATTACH'`
	`...`
	`  MTE_CLASS = 'MSC_ATTRIBUTE_CLASS'`
	`  MSC_CUSTOMIZING_GROUP = 'MY_MSC_GROUP'`
	
	Implementation help: See *Properties: MTE Classes, Customizing Groups* on page 42.

Table 1: Structural and Organizational Decisions.

Decision Table: Structuring Your Monitor

Design Consideration / Info Requirement	Choices / Specifications

Using Summary MTEs to group objects and attributes: You can structure your monitoring tree as you wish by inserting summaries as organizing elements.
 	
Summaries have no monitoring functionality, but can carry method assignments.
	Labels for structural elements (summaries) for grouping subordinate MTEs (for display – up to 20 char):
	_____
	_____
	
	Position in your monitoring tree:
	At which points in your monitoring tree should summaries appear? What are the parent MTEs?
	
	Visibility: Should a summary be:
	__ Always visible in the alert monitor (operator visibility)
	__ Reserved for higher-level users (administrator visibility)
	__ Reserved for developer (SAP-internal / expert) use.
	
	Sample code:
	`CALL FUNCTION 'SALI_SUM_CREATE_ATTACH'`
	`  EXPORTING`
	`  PARENT_TID = PARENT_TID`
	`  " TID (handle) of MTE under which`
	`  " the new monitoring summary should`
	`  " appear.`
	`  SUM_NAME = MY_SUMMARY`
	`  " Label of the new summary for`
	`  " display in the alert monitor.`
	`  VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR`
	`  " Specify level for display in alert monitor:`
	`  " al_visible_operator: Always visible.`
	`  " al_visible_expert: Only visible in higher`
	`  "  Administrator/Analysis view`
	`  " al_visible_developer: Only visible in Expert`
	`  "  Analysis (SAP-internal) view`
	
	Implementation help: See *Structuring Your Monitoring Tree: Summaries and References* on page 45.

Design Consideration / Info Requirement	Choices / Specifications

Using References to Link MTEs into your monitoring tree: References let you display values and alerts from another pre-existing MTE without redundant programming, data collection, or alert generation.	**Which MTEs should be referenced in your monitoring tree:** (Which information that already exists in the monitoring architecture is essential for the monitor you are building?
	_____
	_____
	_____
	
	Position in the monitoring tree (parent MTEs for the referenced MTEs):
	_____
	_____
	_____
	
	Labels for the referenced MTEs:
	_____
	_____
	_____
	
	Sample code:
	
	`CALL FUNCTION 'SALI_REFERENCE_CREATE_ATTACH'`
	`  EXPORTING`
	`  PARENT_TID = PARENT_TID`
	`  " TID (handle) of MTE under which`
	`  " the new referenced MTEs should`
	`  " appear.`
	`  MTE_NAME = MY_REFERENCE`
	`  " Name of the reference -- NOT displayed in`
	`  " the alert monitor.`
	`  VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR`
	`  " Specify level for display in alert monitor:`
	`  " al_visible_operator: Always visible.`
	`  " al_visible_expert: Only visible in higher`
	`  "   Administrator/Analysis view`
	`  " al_visible_developer: Only visible in Expert`
	`  "   Analysis (SAP-internal) view`
	`  REFERENCE_TEXT     = Label to show in the tree`
	`  REFERENCE_TID      = Root TID of the`
	`  referenced tree`
	
	Implementation help: See *Structuring Your Monitoring Tree: Summaries and References* on page 45.

Table 2: Structuring a Monitor with Summary and Reference MTEs.

Decision Table: Monitoring Objects

Design Consideration / Info Requirement	Choices / Specifications

Monitoring object specifications: Your component appears as a monitoring object in the alert monitor. You must name the object and specify its location in the monitoring tree.	**Name of your component** as a monitoring object (for display – up to 20 char): _____
	
	Position in the monitoring tree:
	__ In the standard monitoring tree (for example, under each server)
	__ In its own monitoring context (separate monitoring tree directly under each R/3 System)
	Visibility: Should your object be:
	__ Always visible in the alert monitor (operator visibility)
	__ Reserved for higher-level users (administrator visibility)
	__ Reserved for developer (SAP-internal/expert) use.
	
	Sample code:
	`CALL FUNCTION 'SALI_MO_CREATE_ATTACH'`
	`  EXPORTING`
	`  PARENT_TID = PARENT_TID`
	`  " TID (handle) of MTE under which`
	`  " the new monitoring object should`
	`  " appear.`
	`  MO_NAME = SAMPLE_OBJECT_NAME`
	`  " Name of the new monitoring object for`
	`  " display in the alert monitor.`
	`  VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR`
	`  " Specify level for display in alert monitor:`
	`  " al_visible_operator: Always visible.`
	`  " al_visible_expert: Only visible in higher`
	`  "   Administrator/Analysis view`
	`  " al_visible_developer: Only visible in Expert`
	`  "   Analysis (SAP-internal) view`
	
	Implementation help: See *Defining a Monitoring Objecton* page 45and *Visibility in the Alert Monitor* on page 41. If your object should have its own context, see *Creating Your Own Monitoring Context* on page 56.

Table 3: Monitoring Objects.

Decision Table: Monitoring Attributes for Performance Values

Design Consideration / Info Requirement	Choices / Specifications

Monitoring attribute for performance: Definition.	**Performance value** (fill out once per monitored value):
	
	Name of monitoring attribute (reported value, for display in monitor): _____
	Parent MTE (monitoring object) in the monitoring tree: _____
	Unit (up to 4 characters): _____
	Severity (importance of component being monitored - 0 low - 255 high): _____
	
	Visibility: Should the monitoring attribute be:
	__ Always visible in the alert monitor (operator visibility)
	__ Reserved for higher-level users (administrator visibility)
	__ Reserved for developer (SAP-internal) use.
	
	Sample code:
	
	`CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'`
	`...`
	`  PARENT_TID  = MO_TID`
	`  " TID (handle) of the monitoring object`
	`  " under which the attribute is to appear.`
	`  MTE_NAME    = "SAMPLE_ATTRIBUTE_NAME"`
	`  " Name for display in the alert monitor`
	`  MTE_SEVERITY  = 50`
	`  " Value from 0 (minor) to 255 (severe) that`
	`  " shows how important an alert in the MA is.`
	`  " Ex: Too many users is not critical.`
	`  " Failed updates are.`
	`  MTE_VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR`
	`  " See "Monitoring Object" above.`
	`  PERF_UNIT_TO_DISPLAY = 'Usrs'`
	`  " Unit of performance attr. for display in`
	`  " monitor. Up to 4 characters.`
	`  PERF_DECIMALS        = 0`
	`  " Decimal places to display`
	`  MTE_SECONDS_WARMUPTIME  = 0`
	`  " Ignore alerts for this`
	`  " time period (sec.) during warm-up.`
	`  " Value 0 inactivates.`
	
	Implementation help: See *Defining a Monitoring Attribute* on page 47.

Design Consideration / Info Requirement	**Choices / Specifications**
Monitoring attribute for performance: Alert thresholds: When should the monitoring architecture generate alerts based on performance values?	Trigger alerts when **ABOVE | BELOW** thresholds? Evaluate performance **VALUES** or **FREQUENCY/MIN**. of reports? **Keep alerts:** • To limit of ___ (default 10) • Discard in order of __ Age (Ignore KEEP_ALMAX. Keep as many as possible in chronological order, use as much storage as is available) __ Keep oldest __ Keep newest __ Keep most severe __ Keep those that have the same status as the monitoring attribute **Should alerts be ignored during a warm-up period?** __ Yes __ No Threshold for **yellow** alert (warning): ____ units Threshold for **red** alert (problem): ___ units Threshold for resetting status from **red to yellow**: ____ Threshold for resetting status from **yellow to green**: ____ (Set these to "lower" values than the alert thresholds to prevent "flickering" between statuses). Sample Code: **CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'** **...** **MTE_KEEPALTYPE = AL_KEEP_ALL** " Keep alerts? Default: keep all to limit of " available storage. Others: " al_keep_oldest " al_keep_newest " al_keep_highest " al_keep_slave **MTE_KEEPALMAX = 10** " Keep a maximum of 10 alerts. Discard excess " according to keepaltype (except for setting " al_keep_all). **MTE_SECONDS_WARMUPTIME = 0** " Ignore alerts for this time period (sec.) " during warm-up. **PERF_SUBTYPE = AL_STD_NO_SUBCLASS** " Monitor behavior: check values of reported " data. Alternate:

	" al_std_perf_freq_1_minute check frequency of " msgs. **PERF_RELEVANT_VALUE = AL_PERF_RV_LAST** " Evaluate most recent data to trigger alert. **PERF_THRESHOLD_DIRECTION = AL_THRESHDIR_ABOVE** " Trigger alert when a threshold is exceeded. " _below, when threshold is not reached. **PERF_THRESHOLD_GREEN_TO_YELLOW = 50** " Trigger yellow alert **PERF_THRESHOLD_YELLOW_TO_RED = 100** " Trigger red alert **PERF_THRESHOLD_YELLOW_TO_GREEN = 40** " Return condition to green from yellow **PERF_THRESHOLD_RED_TO_YELLOW = 90** " Return condition from red to yellow. " Return values lower to prevent 'flickering'. Implementation help: See *Defining a Monitoring Attribute* on page 47and *Setting Alert Thresholds for a Performance Monitoring Attribute* on page 55.

Table 4: Monitoring Attributes for Performance Values.

Decision Table: Monitoring Attributes for Status Messages

Design Consideration / Info Requirement	**Choices / Specifications**
Monitoring attribute for status messages: Definition	**Status message** (fill out once per status attribute): **Name of monitoring attribute** (component or type of message, for display in monitor): _____ **Parent MTE** (monitoring object) in the monitoring tree: _____ **Severity** (importance of component being monitored - 0 low - 255 high): _____ **Visibility**: Should the monitoring attribute be: __ Always visible in the alert monitor (operator visibility) __ Reserved for higher-level users (administrator visibility) __ Reserved for developer (SAP-internal/expert) use. **Keep alerts**: __ To limit of ___ (default 10) __ Discard in order of __ Age (Ignore KEEP_ALMAX. Keep as many as possible in chronological order, using as much storage as is available) __ Keep oldest __ Keep newest __ Keep most severe __ Keep those that have the same status as the monitoring attribute Sample code: **CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'** **...** **PARENT_TID = MO_TID** " TID (handle) of the monitoring object " under which the attribute is to appear. **MTE_NAME = "SAMPLE_ATTRIBUTE_NAME"** " Name for display in the alert monitor **MTE_SEVERITY = 50** " Value from 0 (minor) to 255 (severe) that " shows how important an alert in the MA is. " Ex: Too many users is not critical. " Failed updates are. **MTE_VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR** " See above. **MTE_KEEPALTYPE = AL_KEEP_ALL** " Keep alerts? Default: keep all to " limit of available storage. " Others: " al_keep_oldest " al_keep_newest " al_keep_highest " al_keep_slave

	MTE_KEEPALMAX = 10 " Keep a maximum of 10 alerts. Discard " excess according to keepaltype. **MTE_SECONDS_TIL_COLLECTINGTOOL = 0** " How often should alert monitor " start collection tool? Here, " our program reports messages " as needed. The value 0 tells " the alert monitor not to start " the collection tool. **MTE_SECONDS_UNTIL_SET_INACTIVE = 0** " Is element active? Set to 0 (off) " because our program reports status " at unpredictable intervals. **MTE_SECONDS_WARMUPTIME = 0** " Ignore alerts for this time period " (sec.) during warm-up. Implementation help: See *Defining a Monitoring Attribute* on page 47.

Design Consideration / Info Requirement	**Choices / Specifications**

Monitoring attribute for status messages: How and when should messages produce alerts?	**Should basic condition of attribute be green or white:**
	__ Green. If no message or unresolved alert is pending, then attribute is green (everything OK).
	__ White. If no message or unresolved alert, then attribute is white (attribute inactive, no values available).
	
	Should alerts be generated for status messages:
	__ Always
	__ Only when the criticality changes (a higher criticality is reported)
	__ Only when a new message (new number), criticality or severity is reported
	__ Never.
	
	Should alert criticality be shifted for status messages:
	__ No, accept criticality as reported
	__ Shift all red alerts to yellow (no red alerts allowed)
	__ Shift all yellow alerts to red (all alerts are red)
	__ Shift each alert down one level in criticality (no red alerts -- shift red alerts to yellow; ignore yellow alerts -- shift yellow alerts to green (no alert))
	
	Sample code:
	
	`CALL FUNCTION 'SALI_SMES_CREATE_ATTACH'`
	`  SMES_SUBTYPE = AL_STD_NO_SUBCLASS`
	`  " Default value AL_STD_NO_SUBCLASS`
	`  " Monitor behavior: No alert implies`
	`  " value "Green". New or cleared`
	`  " attribute is green in the monitor.`
	`  " Alternate:`
	`  " al_std_smes_green_only_explicit`
	`  " No alert: Attribute is inactive,`
	`  " color white in monitor. Attribute`
	`  " is set to green only by reporting`
	`  " of message with green criticality.`
	`  SMES_ALERT_MODE = AL_SMSG_ALMODE_ALWAYS`
	`  " Modify alert-triggering behavior.`
	`  " Here:`
	`  " Always trigger an alert if a message`
	`  " with yellow or red criticality is`
	`  " reported (See also RZ21).`
	`  " Alternatives:`
	`  " al_smsg_almode_value_chg`
	`  " Trigger alert only on change in`
	`  " message criticality (e.g. yellow`
	`  " to red) or if no alert exists yet.`
	`  " al_smsg_almode_msg_chg`
	`  " Trigger alert only on change in`
	`  " message reported (new msg. Number)`
	`  " al_smsg_almode_never`
	`  " Never trigger an alert, only`
	`  " report (and display) messages.`
	`  SMES_ALERT_SHIFT = AL_SMSG_ALSHIFT_UNCHG`
	`  " Shift criticality of alerts. Here:`

	" Use the criticality of the reported " message to determine the criticality " of the alert that is triggered. " Alternatives: " al_smsg_alshift_r_as_y " No red alerts. Red becomes yellow. " al_smsg_alshift_y_as_r " No yellow alerts. Yellow becomes " red. " al_smsg_alshift_ray_yag " Downgrade all alerts. Red becomes " yellow; yellow becomes green. Implementation help: See *Controlling Whether and How Messages Trigger Alerts: Status and Message Container Attributes* on page 48.

Design Consideration / Info Requirement	Choices / Specifications

Monitoring attribute for status messages: Alert criticality of messages, and message number	**Criticality** of the message (fill out once per message that you may report): _____ (Green, yellow, or red alert)
	
	`CALL FUNCTION 'SALI_SMES_REPORT_T100_MESSAGE'`
	`...`
	`  MSGVALUE = AL_VAL_YELLOW`
	`  " Criticality of the message:`
	`  " al_val_green: Success message.`
	`  " al_val_yellow: Warning message.`
	`  " al_val_red: Error/Problem message.`
	`  MSGID = 'RA'`
	`  MSGNO = '094'`
	`  " Message ID and number from table`
	`  " T100, and values for message variables....`
	`  MSGARG1 = VAL_1`
	`  ARGTYPE1 = TYPE_1`
	
	Implementation help: See *Messages in the Alert Monitor* on page 41, *Controlling Whether and How Messages Trigger Alerts* on page 48, and *Controlling Alert Ranking* on page 50.

Table 5: Monitoring Attributes for Status Messages.

Decision Table: Monitoring Attributes for Message Log Attributes

Design Consideration / Info Requirement	Choices / Specifications
Monitoring attribute for logs/traces (message containers): Definition When should the monitoring architecture generate alerts based on log attributes that you report?	**Basic definition:** See the table entries for status attributes, above, for decision table and sample code. Only special options are shown separately here. Special options for log attributes: **Log source:** __ The alert monitor should implement a log or trace for my component (cached log). __ (Option not yet available). I wish to have the alert monitor access my own external log to display messages. **Message to display in monitor:** __ Last message added to log attribute. __ Most important message currently available (highest criticality and severity in log). __ Most important message in the last ___ minutes. **CALL FUNCTION SALI_MSC_CREATE_ATTACH** ... **MSC_SUBTYPE = AL_STD_MSC_CACHE** " Default value " Behavior: Monitoring architecture " provides log functionality. Messages " are held in an internal cache " provided by the monitoring " architecture. See log type docu. " Alternates: " al_std_msc_syslog " The R/3 system log provides the " message log functionality. Only " alerts are held by the monitoring " architecture, message are read from " the system log for display. " al_std_msc_external " A log facility external to the " monitoring architecture provides " the log functionality (not the " R/3 system log). You must provide " the function call for retrieving " messages from the log for display. " At this time, only "cache" is allowed. **MSC_ACTUAL_MSG_MODE = AL_MSC_VAL_MODE_LAST** " Default " Determines which message should be " displayed as the current status msg " in the alert monitor. Here, the " most recent message is displayed. " Alternatives: " al_td_msc_val_mode_highalrt: Display " the most important alert. " al_td_msc_val_mode_worst_since: Display " most important alert (if any) since " during the last X minutes. X is " specified in msc_actual_msg_time. **MSC_ACTUAL_MSG_TIME = 20**

	" Omit unless mode is "worst_since". " Then specifies "find worst message " in the last actual_msg_time minutes." Implementation help: See *Defining a Monitoring Attribute* on page 47.

Design Consideration / Info Requirement	**Choices / Specifications**
Monitoring attribute for logs/traces: Screening messages before triggering alerts and message ID	As also with status attributes, you can **control whether messages reported to a log attribute trigger alerts**. **When should a message trigger an alert?** __ Always. Use the criticality of the message to determine whether to trigger an alert. Values: AL_VAL_GREEN, Severity 255 __ Only if both of these conditions are met: __ Message severity (0 unimportant -- 255 important) __ Message criticality (yellow or red) __ Never. Only display messages. Values: AL_VAL_RED, severity 255. **CALL FUNCTION 'SALI_MSC_CREATE_ATTACH'** ... **MSC_RAISE_VALUE = AL_VAL_GREEN** " Default " With msc_raise_severity, determines " when alerts can be triggered by " messages in message logs. "green" " lets all messages with criticality " yellow or red trigger alerts. " 'al_value_yellow' would allow only " messages with criticality yellow or red " & severity higher than msc_raise_sever. " to trigger alerts. Values defined " in rsalexti. **MSC_RAISE_SEVERITY = 255** " Default " With msc_raise_value, determines " when alerts can be triggered by " messages in message logs. Here, " only messages with more than " green/255 trigger alerts (any " yellow or red, no green messages). " See also message screening docu. **What criticality and severity should a message have?** (Fill out for each message that may be added to a log). The values are probably best held in the module that contains SALI_MSC_REPORT_T100_MESSAGE. You can then assign the correct value and severity to each message as it is issued. The alternative: creating a repository for these attributes of your messages. **CALL FUNCTION 'SALI_MSC_REPORT_T100_MESSAGE'** ... **VALUE = MSG_VAL** " Criticality of the message: " al_val_green: Success message. " al_val_yellow: Warning message. " al_val_red: Error/Problem message. **SEVERITY = 50** " Default is 0 " Importance of the message for preliminary

	" screening in the message container, if " desired. This severity makes it possible, " with "value" to restrict generation of " alerts. See also MSC_RAISE_VALUE and " MSC_RAISE_SEVERITY in " SALI_MSC_CREATE_ATTACH. **MSGID = 'RA'** **MSGNO = '093'** " Message ID and number from table " T100. **MSGARG1 = VAL_1** **ARGTYPE1 = TYPE_1** " Parameter type and value for the " optional parameters in T100 " messages. Implementation help: See *Messages in the Alert Monitor* on page 41 and *Controlling Whether and How Messages Trigger Alerts* on page 48.

Table 6: Monitoring Attributes for Message Logs.

Decision Table: Monitoring Attributes for Text Attributes

Design Consideration / Info Requirement	Choices / Specifications

Text attribute for monitoring object: Text attributes are for reporting infrequently-changed information for which no alert functionality is required. Example in system: System and Server Configuration trees.
 	
The text to display is passed with the CREATE_ATTACH call; there is no separate function module for setting a text.
	Name of monitoring attribute (reported text, for display in monitor): _____
	Parent MTE (monitoring object) in the monitoring tree: _____
	
	Text to display (up to 256 characters): _____
	
	Visibility: Should the text attribute be:
	__ Always visible in the alert monitor (operator visibility)
	__ Reserved for higher-level users (administrator visibility)
	__ Reserved for developer (SAP-internal) use.
	
	Sample code:
	
	`CALL FUNCTION 'SALI_MO_ASSIGN_TEXTATTRIBUTE'`
	`...`
	`  PARENT_TID  = MO_TID`
	`  " TID (handle) of the monitoring object`
	`  " under which the attribute is to appear.`
	`  ATTR_NAME    = "TEXT_ATTRIBUTE_NAME"`
	`  " Name for display in the alert monitor`
	`  ATTR_TEXT  = 'Version 1.0.2.1.3'`
	`  " Up to 256 characters of text that are to`
	`  " be displayed with this text atttribute.`
	`  " Text is modifiable with next call to`
	`  " function module.`
	`  MTE_VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR`
	`  " See "Monitoring Object" above.`
	
	Implementation help: See.

Table 7: Text Attributes for Monitoring Objects.

Implementation in Detail

Background Information: Monitoring Objects, Monitoring Attributes, and Program Structure

Before you start implementing your data supplier, you need a little bit of background information on the monitoring architecture.

In the monitoring architecture and alert monitor, the components to be monitored are represented as "**monitoring objects**." A monitoring object might be, for example, the R/3 update system, batch-input sessions, or a component in an R/3 application.

You cannot report data -- such as performance values or error messages -- directly to a monitoring object. Instead, the monitoring architecture asks you to create a **monitoring attribute** for each aspect of a monitoring object that you wish to monitor.

In the example below, you wish to monitor an application component of some sort, perhaps a business process. You wish to monitor this component for performance (the throughput of completed transactions) and for error messages.

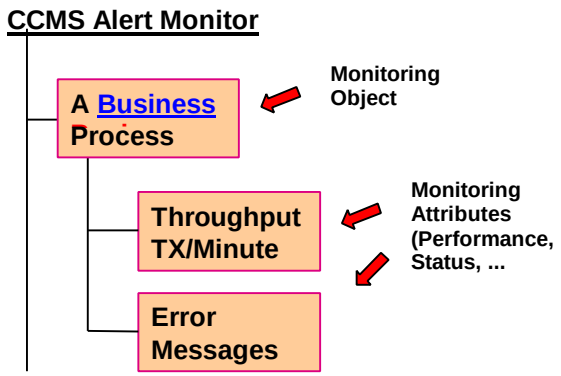

Figure 2. The Monitoring Tree and Monitoring Tree Elements.

The business process is the monitoring object in the alert monitor. Each type of data that is to be monitored for the business process is represented as a monitoring attribute.

Taken together, the hierarchy of objects and attributes is called the **monitoring tree**. Each major branch in the tree (all of a set of objects and attributes that belong to a hierarchy that appears directly under an R/3 System) is called a **monitoring context**. R/3 instances are common examples of monitoring contexts. The monitoring context is always the second element in the name of an MTE.

Another bit of terminology: All of the objects and attributes in the tree are known collectively as **monitoring tree elements** (MTEs). And, there are additional types of MTEs besides objects and attributes: most importantly, virtual and summary MTEs. Virtual and summary MTEs have no function other than to let you structure the tree.

Program Structure in Overview: Release 4.5x

The object/attribute structure determines the general plan of a data supplier as well. The first step in a data supplier is always decide where to add your MTEs to the monitoring tree. The second step is

to create a new monitoring object or attach to an existing one (function module SALI_MO_CREATE_ATTACH). The third step is then to create or attach to monitoring attributes (various function modules, depending upon the type of data to be reported): Finally, the data supplier program gathers its data and passes it to the monitoring architecture for evaluation and display. Typically, all of these tasks are carried out in a single program.

The illustration below shows the sequence of function module calls in Release 4.5A of R/3.

```
Report datasupplier_1...
* Step 1: Decide where to add MTEs to the monitoring tree
call function SALI_MT_GET_TID_BY_NAME    " Get the TID (handle) of an existing MTE
                                         " in the monitoring tree
*  or
call function SALI_MC_CREATE_ATTACH      " Create your own monitoring context

* Step 2: Create one or more monitoring objects (which stand for components)
call function SALI_MO_CREATE_ATTACH...   " Create or attach to
                                         " monitoring object

* Step 3: Create one or more monitoring attributes (which stand for the types
* of information that you wish to report)
call function SALI_PERF_CREATE_ATTACH .  " Create (for example)
                                         " a performance monitoring
                                         " attribute, and/or
call function SALI_SMES_CREATE_ATTACH    " Create (for example) a
                                         " status attribute for reporting
                                         " single status messages, and/or
call function SALI_MSC_CREATE_ATTACH     " Create a log attribute for
                                         " reporting messages from a log
                                         " or trace in context

* Step 4: Gather performance data (application-specific code) and then

* report the data that you have collected to the monitoring architecture
call function SALI_PERF_REPORT_VALUE     " Report performance value to
                                         " monitoring architecture and/or
call function SALI_SMES_REPORT_T100_MESSAGE " Report a status message to a
                                         " status monitoring attribute
call function SALI_MSC_REPORT_T100_MESSAGE  " Report a message to a message
                                         " container (log or trace)
```

General Structure of a Data Supplier: Release 4.5A

Note: You can find a complete explanation of the monitoring architecture, its terminology and concepts, in the monitoring architecture white paper. You can access this paper in the *System Management/CCMS/Media Center* branch of the SAPnet *Information* service.

Program Structure in Overview: Release 4.0B

The programming interface was changed between Release 4.0B and Release 4.5A. The 4.5A interface combines the 4.0B calls into fewer function modules, and the interface is therefore easier to use. The 4.5A interface is available as a hot package for installation on Release 4.0B R/3 Systems, as an alternative to using the Release 4.0B interface.

This document concentrates on the 4.5A interface. However, you can still use this document to help you write data suppliers with the 4.0B interface. The function module calls were changed between 4.0B and 4.5A, but the information that is passed in parameters was not changed. The table in *Mapping 4.0B to 4.5A Function Modules*, below, shows the Release 4.0B function modules that you will need to use. If you have questions about Release 4.0B parameters, see the documentation of the corresponding 4.5A function module.

Here is the general structure of a data supplier in Release 4.0B.

```
Report datasupplier_1...

* Create monitoring object

call function SALI_MO_CREATE_ATTACH...   " Create or attach to
                                         " monitoring object
call function SALI_TOOL_PRESET           " Assign tools to monitoring object
                                         " created above

* Create and customize monitoring attribute...

call function SALI_MA_CREATE_ATTACH      " Create or attach to a monitoring
                                         " attribute
call function SALI_MA_GENER_CUSTOMIZE_PRESET " Assign general properties to
                                         " monitoring attribute created above
                                         " Note:  Specify MA customizing group
                                         " of SALI_MA_CREATE_ATTACH as
                                         " MT Class -- see note below
call function SALI_TOOL_PRESET           " Assign tools to monitoring attribute
call function SALI_SMES_CUSTOMIZE_PRESET " Set customizing for status attribute
                                         " (if applicable)
                                         " Note:  Specify MA customizing group
                                         " of SALI_MA_CREATE_ATTACH as
                                         " MT Class -- see note below
call function SALI_PERF_CUSTOMIZE_PRESET " Set customizing for performance
                                         " (if applicable)
                                         " Note:  Specify MA customizing group
                                         " of SALI_MA_CREATE_ATTACH as
                                         " MT Class -- see note below

* Gather performance data (application-specific code)

call function SALI_PERF_REPORT_VALUE     " Report value to monitoring
                                         " architecture and/or
call function SALI_REPORT_T100_MESSAGE   " Report status message for
                                         " status (single message) attribute
```

General Structure of a Data Supplier: Release 4.0B

Note: To simplify management of MTE classes and customizing groups in Release 4.0B, SAP recommends that you use the same name for the MTE class and customizing group of a monitoring object or monitoring attribute.

Example: A monitoring object named CPU could have customizing group CPU_CUST and MTE class CPU_CUST. This naming convention means that you do not have to worry about correctly passing the MTE class and customizing group between function module calls.

Mapping 4.0B to 4.5A Function Modules

For This 4.0B Function Module...	**See the documentation for this 4.5A Function Module**
SALI_MO_CREATE_ATTACH	SALI_MO_CREATE_ATTACH
SALI_MA_CREATE_ATTACH	SALI_PERF_CREATE_ATTACH or SALI_SMES_CREATE_ATTACH
SALI_MA_GENER_CUSTOMIZE_PRESET	SALI_PERF_CREATE_ATTACH or SALI_SMES_CREATE_ATTACH

SALI_SMES_CUSTOMIZE_PRESET	SALI_SMES_CREATE_ATTACH
SALI_PERF_CUSTOMIZE_PRESET	SALI_PERF_CREATE_ATTACH
SALI_TOOL_PRESET	SALI_MO_CREATE_ATTACH or SALI_MA_CREATE_ATTACH
SALI_PERF_REPORT_VALUE	SALI_PERF_REPORT_VALUE
SALI_SMES_REPORT_T100_MESSAGE	SALI_SMES_REPORT_T100_MESSAGE

Note: Log attributes (message containers) were not available in Release 4.0B.

Type of Information to Provide: Performance, Status Messages, Log/Trace, Text

What type of information do you wish to report into the alert monitor? The information type determines which function modules from the monitoring architecture you will need to use in your data supplier:

Performance	**1. SALI_MO_CREATE_ATTACH** to register your component as a "monitoring object" with the monitoring architecture ("plug in" to the monitoring architecture). **2. SALI_PERF_CREATE_ATTACH** to register a "monitoring attribute" with the monitoring architecture ("plug in" to the monitoring architecture). **3. SALI_PERF_REPORT_VALUE** to report values for a single performance attribute or **SALI_PERF_REPORT_TABLE_OF_VAL** to report values for multiple performance attributes simultaneously. Sample program: RSDSSMPL_PERFORMANCE
Status (single message)	**1. SALI_MO_CREATE_ATTACH** to register your component as a "monitoring object" with the monitoring architecture ("plug in" to the monitoring architecture). **2. SALI_SMES_CREATE_ATTACH** to create a status attribute into which messages can be reported. **3. SALI_SMES_REPORT_T100_MESSAGE** to report a message. Sample program: RSDSSMPL_STATUS
Log attribute (message container)	**1. SALI_MO_CREATE_ATTACH** to register your component as a "monitoring object" with the monitoring architecture ("plug in" to the monitoring architecture). **2. SALI_MSC_CREATE_ATTACH ATTACH** to create a log attribute into

	which messages can be reported. **3. SALI_MSC_REPORT_T100_MESSAGE** to add a message to a log attribute (message container). Sample program: RSDSSMPL_LOG
Text attribute (information only)	**1. SALI_MO_CREATE_ATTACH** to register your component as a "monitoring object" with the monitoring architecture ("plug in" to the monitoring architecture). **2. SALI_MO_TEXTATTRIBUTE** to create an attribute for reporting static, textual information (such as the name of the administrator responsible for a system), together with the information that is to be displayed.

You can report as many different types of information for a particular monitoring object as you wish.

In general, you will want to implement a separate data supplier program for each of the types of data that you wish to report. The reason: the data suppliers start for different reasons. A data supplier that gathers performance data will probably start periodically. A data supplier that supplies a message or adds messages to a log attribute, by contrast, is event-driven. It must start when a message occurs.

It is possible, of course, to supply all types of monitoring data from a single program, but your data supplier would then need intelligence to decide what to do. It is simpler to build separate, simple, data-type specific data suppliers.

Scope of Your Data Supplier

A data supplier, and by implication, the monitoring objects and attributes that it creates, can be implemented as either:

- **Local to an R/3 instance**. In this case, an object and attribute represent a component that pertains to a particular R/3 instance or server. Relevant performance values or messages can be collected only on a per-instance basis. The data supplier must run separately on each R/3 instance or other component that is to be monitored.

  Example: R/3 dialog response time is a server-local performance attribute in the alert monitor. It is possible, but not useful, to measure system-wide dialog performance, because unsatisfactory performance must always be analyzed at the R/3-instance level. The dialog response time attribute appears in the monitoring context of each R/3 instance in a system.

- **Global in an R/3 System**. In this case, an object and attribute pertain to a component that is present only once in an R/3 System. Relevant performance values or messages can be collected only on a system-wide basis. Only a single exemplar of the data supplier may run in an R/3 System.

  Example: Monitoring for aborted background jobs is a system-global activity. A single instance of the data supplier can report for the entire R/3 System on the occurrence of abnormal job terminations. It is not possible to monitor for this problem on each separate instance of the R/3 System.

Implications for Storage and Display of Data

Your choice of instance-local or system-global implementation should reflect the type of data that you are collecting. Create data suppliers that run on each instance if you need to collect data that is local to each R/3 application server. Create a system-global application server if you wish to report data that can be collected once for the whole system, such as information from the R/3 database.

It is not possible to collect data at multiple servers and then report it into a single 'system-global' context. This is because the monitoring architecture cannot execute RFC calls from a passive collection method (a data supplier that is started automatically by the monitoring architecture).

Do not worry about packaging server-specific and system-global data that belongs together into the same monitoring structure or even into the same data supplier program. The monitoring architecture distinguishes between the storage strategy and the display strategy for information. In your data supplier, you should follow the requirements of efficient information gathering and storage. You can provide the desired view onto the data by pre-defining and transporting an appropriate monitor in transaction RZ20. With a monitor definition, you can gather any desired set of monitoring data into a single view, a single monitor, regardless of where and how the data is stored.

Implementation

To implement an instance-local data supplier, you will need to do the following:

1. In your data supplier, insert your monitoring object and its attributes into the monitoring context of each R/3 instance.

   The sample program RSDSSMPL_PERFORMANCE shows how to do this with the function modules SALI_MT_GET_TID_BY_NAME and SALI_MO_CREATE_ATTACH.

2. Ensure that your data supplier is started on each server in your system.

   If the data supplier is to be defined as a collection method (a passive data supplier), then you can use the method definition to ensure that the program runs on each server or R/3 instance.

   For example, you can define the method in the monitoring architecture (transaction RZ21) and specify that:

   - it is to be started automatically on each R/3 instance as the instance is started; and that
   - it should run in the server to which the monitoring object or attribute pertains.

The method will then be started automatically as each R/3 instance in the system is started. Depending upon your other specifications, the method will thereafter run periodically in each server.

To implement a system-global data supplier, you will need to do the following:

1. In your data supplier, use SALI_MC_CREATE_ATTACH to create/attach to your own monitoring context.

   This monitoring context is independent of R/3 instances within your current R/3 System. You can insert it at the same level as the instances in the monitoring tree.

   Optionally, you can also use the FORCE parameter of SALI_MC_CREATE_ATTACH to have the monitoring architecture automatically move your data supplier if the R/3 instance in which it is running is stopped. The monitoring architecture then re-starts the program on another instance in the same R/3 System. See *Creating Your Own Monitoring Context* on page 56 for more information.

2. In your data supplier, create/attach your monitoring objects and attributes in the newly created monitoring context.

You use the same function module calls as in a server-local data supplier. You need only create/attach your monitoring objects in the TID that you obtained with the SALI_MC_CREATE_ATTACH call. (A TID is the identifier for a monitoring tree element (MTE) in the alert monitor).

3. Ensure that your data supplier is started only once per R/3 System.

   For example, you can define the data supplier as a collection method in the monitoring architecture (transaction RZ21). In the definition, you can specify, for example, that the collection method:

   - is to be started only on the system's central instance, and
   - is to be started automatically at R/3 instance start (run method at start of monitoring segment).

   The collection method will then run only on the central instance of an R/3 System.

   Alternatively, you can specify in the method definition that the method:

   - Is to be started on any available server, and
   - Is to be started automatically at server start.

   Because you have assigned your MTEs to a unique monitoring context, the method will be started successfully only once, on the first instance to start in the R/3 System. The monitoring architecture will abort all starts of the method on other R/3 instances.

   Should you choose the latter method, you can have your data supplier exit gracefully by catching the WRONG_SEGMENT exception from the first CREATE_ATTACH call in your program:

```
Call function 'SALI_MO_CREATE_ATTACH'
         ....
         Exceptions
                ....
                WRONG_SEGMENT = 5
                ....

if sy-subrc <> 0.
         If sy-subrc = 5.
                Exit.
         Endif.
Endif.
```

WRONG_SEGMENT signals that another instance of the data supplier has already run in another server (another monitoring segment).

Specifying Methods to Use in Your Data Supplier

Your method specifications can appear either when you create the monitoring object or when you create the monitoring attribute (see *Dispatching Your Data Supplier* on page 35). In both cases, the parameters are the same. For example:

```
call function 'SALI_MO_CREATE_ATTACH'   " Create monitoring object
                                        "  or
call function 'SALI_PERF_CREATE_ATTACH' " Create Performance monitoring
                                        " attribute, or status attribute, or
                                        " log attribute...
....   ...
       tool_collecting        = 'user_check'
                              " Logical name of program
                              " to run to collect data,
```

```
                          " defined in RZ21. The program that
                          " you are writing here is usually defined
                          " with RZ21 as the collection method. (Exception:
                          " You are writing an "active" data supplier.
tool_onalert           = 'tell_admin_too_many_users'
                          " Logical name of program
                          " to start if an alert is
                          " triggered.
tool_analyze           = 'active_users'
                          " Tool for analyzing alerts
                          " or current condition.
```

Each monitoring object and attribute can have its own method or complete set of methods. Attributes can also inherit their methods from the parent monitoring object.

Note that the methods must be defined in the monitoring architecture with transaction RZ21. Enter the logical names of the methods, as defined in RZ21, in the function call.

Standard methods: The CCMS delivers many pre-defined methods with the R/3 System. The names of such methods in RZ21 always begin with the string **CCMS_**. If the transaction, report, or external command that you need is defined with a CCMS_ name, then you can be assured that it will be available in customer systems at or above the development release level (unless the customer has deleted the method definition). You can therefore use such method names in your collection method.

Dispatching Your Data Supplier

A data supplier program needs to run repeatedly. If it is reporting performance data to the alert monitor, then it probably also needs to run regularly. If it is reporting a message, then it needs to start whenever a relevant message is issued.

The alert monitor offers functionality for restarting a data supplier as a "collection method", or "passive data supplier". When you create a monitoring attribute, you need to specify whether and how this functionality should be used with your program. (See also for more details *Restart Functionality*, later in this section.)

Here are the decisions you need to make:

- First, **should the alert monitor start your data supplier?** If yes, then it is a passive data supplier, or collection method. If no, then it is an active data supplier. In this case, it must be started by your application as it is needed. And the data supplier need not be defined as a collection method in the monitoring architecture.

  Here are your options:

  - If your collection method **runs quickly**, then have the alert monitor start it periodically in a dialog work process. This is then a passive data supplier, or collection method.

    In your collection method program, make these specifications:

    ```
    call function 'SALI_MO_CREATE_ATTACH' (object) or
    'SALI_PERF_CREATE_ATTACH'... (attributes)
    ...
    MTE_SECONDS_TIL_COLLECTINGTOOL = 240 " Time period for restart
                                         " (sec.)
    TOOL_TOOLDISPATCHER            = 'SAP_CCMS_DEFAULT_TD'
                                         " Alert monitor checks every
                                         " 5 min. to see if collection
                                         " tool should run.  When required
    ```

```
                                        " it starts the method in a dialog
                                        " work process.
```

- If your collection method **takes a long time to run** or is performance-intensive, then have the alert monitor start it periodically in a background work process. This is then a passive data supplier, or collection method.

  In your collection method program, make these specifications:

```
call function 'SALI_MO_CREATE_ATTACH' (object) or
'SALI_PERF_CREATE_ATTACH'... (attributes)
...
MTE_SECONDS_TIL_COLLECTINGTOOL = 3600 " Time period for restart
                                      " (sec.)
TOOL_TOOLDISPATCHER           = 'SAP_CCMS_BATCH_DISPATCHER'
                                      " Alert monitor checks every
                                      " hour to see if collection
                                      " method should run, runs method
                                      " in background.
```

- If you are reporting **status messages** or **writing messages to a log attribute**, then your application should probably start the data supplier. In this case, you create an active data supplier.

  It is usually not sensible to use the alert monitor to start data suppliers that register status messages or write to log attributes. You must arrange to start these data suppliers yourself in the event that an important message is issued. You do not need to define such active data suppliers as collection methods in transaction RZ21, unless your users should be able to start the collection method by hand from the alert monitor.

  In your data supplier program, you should also leave out the specifications that pertain to the collection method:

```
call function 'SALI_MO_CREATE_ATTACH' (object) or
'SALI_PERF_CREATE_ATTACH'... (attributes)
...  omit:
     TOOL_COLLECTING
     MTE_SECONDS_TIL_COLLECTINGTOOL
     TOOL_TOOLDISPATCHER
```

  Or, if you need to associate a tool and its collection method, set MTE_SECONDS_TIL_COLLECTINGTOOL to 0.

- **Second, should the alert monitor mark the monitoring object as "inactive"** if the data supplier does not report data within a certain time interval?

  You should probably activate this option if you are defining a performance attribute for which values are to be regularly reported. Something is the matter (or at least the data is old and unreliable) if new values are not reported over a longer time period.

  You should probably deactivate this option if you are reporting status messages or writing to a log attribute. Since such events usually occur irregularly, it makes no sense to deactivate an MTE if a message is not reported within a given time span.

  Code:

```
CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'
. . .
MTE_SECONDS_UNTIL_SET_INACTIVE = 900
                                    " Is element active?
```

```
" If no value is reported
" for 900 seconds, the
" MTE in the monitor goes
" white: Inactive. Set to
" 0 to switch off.
```

Automatic Execution Functionality – Background Information

The monitoring architecture provides functionality for periodically starting a collection method. Further, a method can be run either in a dialog work process (like a normal transaction, but without user interaction). Or a method can be run as a job in the background processing system.

You specify a repeat interval in seconds for the collection method and you specify whether the method should run in dialog or in the background. If the method is to run in a **dialog work process**, then the monitoring architecture then checks every five minutes to see if the repeat interval has elapsed. The monitoring architecture does not guarantee that the collection method will be started exactly at the specified restart interval. Also, the tool can be run a maximum of 12 times per hour.

If the method is to run as a **background job**, then the monitoring architecture checks every hour to see whether the method should run. The method can therefore be run a maximum of 24 times a day.

The automatic execution functionality does not look beyond the boundaries of an R/3 instance. This means that starting server-specific collection methods is uncomplicated: in each instance, the method starts independently. It is not possible for the launching of a method in one instance to prevent the launching of the method in another server, or to interfere with the restart of the method in any way at all.

Where to Specify Dispatching Options

You make basic dispatching specifications -- such as whether a method should be started by the alert monitor -- with the CREATE_ATTACH function module for the monitoring attribute.

You can make additional specifications for dispatching a collection method in the definition of a method (tool) in the alert monitor, with transaction RZ21. See *Defining Methods in Transaction RZ21*Additional Dispatching Options in the Collection Method Definition on page 37 for more information on these special options.

Defining Methods in Transaction RZ21

You can specify data collection, on-alert, and analysis methods when you create or attach to a monitoring attribute. When you do this, you may not directly specify a transaction or report name in your function module call.

Rather, every method must be defined in the monitoring architecture. Your CREATE_ATTACH function module calls may use only the "logical" method names that you specify in these definitions.

You can define methods in the monitoring architecture only with transaction RZ21. This cannot be done automatically (under program control).

Procedure

1. Choose *Tools* → *CCMS* → *Configuration* → *Alert monitor* to start the customizing function.

   Alternative: Enter transaction RZ21.

2. Mark *Method definition* and choose *Display overview*.

3. If you do not see your method in any existing definition, then choose *Create* to define the method.

4. Fill out the method definition screen. You will need to specify the following:

   - A name for the method. This logical name is used to assign the method to a monitoring attribute in a data supplier.

   - When and how the method should be started. (See the F1 field help or the next section for more information.)

   - The type of usage for which the method has been released (*Release* tab). Note that the release mechanism is protected: you must have an administrator authorization (authorization object S_RZL_ADM) to release a method for use in the monitoring architecture.

5. Save the method definition to make it available for use in the alert monitor.

Additional Dispatching Options in the Collection Method Definition

These options must be set in the definition of a collection method in the alert monitor. The definition must be maintained with transaction RZ21.

The options are as follows:

- *Execution* tab, *Run method on* box. Where should a collection method be run when it is periodically started?

  Periodic execution of a collection method is triggered by the MTEs associated with the method. Essentially, the MTE signals to the monitoring architecture that it needs an update – its collection method should be run. The monitoring architecture checks the *Execution* tab, *Run monitor on* box to find out where to run the method.

  This option is important if the collection method program must run on the instance associated with an MTE (the MTE's local instance) or on a particular R/3 instance or host system (RFC destination).

  The choices that you have are as follows:

  - Start the collection method on the server associated with a particular MTE. This is the setting to use for a collection methods that run in dialog. This setting ensures that the collection method starts on the server at which its shared memory segment is located, which is required for dialog methods. Methods that run in the background can access monitoring shared memory segments at other servers.

  - Start the collection method at a specified RFC destination. This can be another R/3 System or an external program. The RFC destination must be defined in the R/3 System (transaction SM59).

  - The alert monitor may start the method on any available server. This is usually the right setting for analysis methods that use data that is available system-wide, (data from the R/3 database).

- *Execution* tab, *Run method for* box. Should a collection method run once for all monitoring attributes that request it? Or should the method be started separately for each attribute?

Multiple monitoring attributes within the context of an R/3 instance can share the same collection method. This is the case with file system collection methods, for example. If the shared method can supply information to all of its monitoring attributes, then you can specify that the method should be started only once when several monitoring attributes "require" that it be run at the same time. Otherwise, the monitoring architecture starts the method separately for each monitoring attribute that needs it.

Example, the file system method may be triggered from several different file system monitoring attributes (disks C:, D:, and E:) at the same time. Since it does not need to run separately for each file system, the method is defined with the option *start once for table of requesting MTEs*. Once the method has been started, its status is switched to "launched." All other requests that the method run are ignored as long as the method is running. Should the method not be able to supply data for each disk, then it would be started separately for each of the monitoring attributes for disks.

- *Control* tab, *Control method execution* box. Here, you can specify that a method should run in a dialog work process, like a transaction, but without user interaction.

  Choose dialog for collection methods that run quickly.

  Choose background execution for a method that runs a long time. Note that background methods can run a maximum of once per hour.

  Choose manual execution for analysis methods or active data suppliers (ones that are not started automatically by the monitoring architecture).

- *Control* tab, *Startup method* box. Should the collection method be started automatically at R/3 System start? Doing so guarantees that the monitoring objects and attributes are visible in the alert monitor right from the start of system operation.

  The alternative: the data supplier is started when it is needed. In this case, the monitoring object and attribute are added dynamically to the alert monitor as required. Your application must start the data supplier in this case, and the data supplier need not be defined as a collection method in the monitoring architecture. It is also sensible to have your application destroy the object and its attributes when they are no longer needed. See Discarding a Monitoring Object and Attributes on page 57.

  Example: A batch-input program could add at runtime an attribute to an existing "data transfer" object. Upon its completion, the report could delete its attribute. The attribute is deleted from the alert monitoring tree as soon as any alerts for the attribute have been "completed."

Number Range Management

Each monitoring tree element (MTE) has a unique ID number from a number range. This number is used in a UID, a unique identification number used internally in the monitoring architecture.

You can have the monitoring architecture assign and manage these ID numbers and number ranges or you can manage the number range yourself. If you let the monitoring architecture manage number ranges, then you have the further choice between permanent and temporary number assignment. (By consequence, this means that you have a choice between permanent and temporary MTEs.)

Recommendation: Have the monitoring architecture assign and manage permanent UIDs for you. Use NUMRANGE = AL_NR_AUTO in CREATE_ATTACH function modules. Permanent UIDs are unique in each R/3 System. MTEs that use such UIDs persist across R/3 System and server restarts, together with all of their data and alerts. **Temporary number ranges:** For initial development and

testing, you may wish to use temporary UIDs (AL_NR_TEMP). Any MTE that uses a temporary number range persists only as long until its server is restarted. When the monitoring segment on the server is re-initialized, the MTE and all data and incomplete alerts associated with it are gone.

You can also use temporary number ranges in production monitoring functionality. Temporary UIDs and MTEs are appropriate whenever there is no need to keep data and alerts across system/server starts.

Code:

```
NUMRANGE = AL_NR_AUTO " Permanent UID, assigned and
  " managed by the monitoring architecture
  " Sample uses the auto number
  " range for MTE handles. Benefits:
  " the monitoring architecture
  " manages the number range; you
  " don't have to; MTEs, data, alerts,
  " MTE classes, customizing groups are persistent
  " Alternative:
  " AL_NR_TEMP -- temporary MTEs, deleted with data
  " and alerts at each system/server restart
  " AL_NR_SYS_PERM -- You manage the number range.
  " The MTEs you create are "permanent" -- they
  " survive a server or system restart.
```

Background Information

The type of number range management you select determines whether your objects and attributes are temporary or permanent:

- **Temporary** UIDs. In this case, objects, attributes, their MTE classes and customizing groups (see *Properties: MTE Classes, Customizing Groups* on page 42) and most importantly, their alerts, persist only as long as the R/3 instance in which they are held keeps running.

  The only exception: If a user has accessed and saved the properties of an object or attribute, then the MTE class and/or customizing group are made permanent. They are saved in the database and persist across instance restarts. This exception applies only to the MTE class or customizing group, not to monitoring objects and attributes.

  When an instance is restarted, the monitoring segment (shared memory, implemented as a memory-mapped file) is lost. With it, everything temporary that was held in monitoring contexts in that segment is deleted. When the server restarts, temporary objects and attributes are re-created as their data suppliers run, and operation continues as normal, with the exception that old alerts and data are gone. If the MTE classes or customizing groups have been saved, then the database versions of these entities are used. (MTE class / customizing group rule: "The database always wins.")

- If you manage your number range or if you use the 'auto' number assignment, then your objects and attributes are **permanent**.

  In automatic numbering, the monitoring architecture manages permanent number range assignments for you. Your MTEs have UIDs that are unique in an R/3 System. The MTEs, together with all of their associated alerts, classes, groups, and data, survive system restarts.

  Partners, customers, and SAP developers should all use automatic number assignment. Owners of MTEs should make themselves known by correctly setting the OWNER parameter in CREATE_ATTACH calls.

Should you wish to manage number assignment yourself, then you (your application) must ensure permanently (across instance and R/3 System restarts) that each object and each attribute receives an ID number within the number range that is assigned to you by the CCMS. The ID number must be unique within the monitoring context in which the object is located.

In an MTE name, the context is always the second element of the name. Example: "spool" is the context of the "MaxWaitTime" monitoring attribute, as shown in the full name of the monitoring attribute: \BCE\Spool\...\SystemWide\MaxWaitTime.

In this case, objects and attributes, their MTE classes and customizing groups, and alerts are all held in the database. You can discard objects and attributes by destroying them, but their MTE classes and customizing groups persist until a user manually deletes them.

You may create permanent objects whose UIDs you wish to manage yourself only in a context that you create. See *Creating Your Own Monitoring Context* on page 56.

If you wish to manage number ranges yourself, then you can get a registered number range from the CCMS. Contact ccms@sap-ag.de. Or you can use ABAP number range functionality to create and use your own number range object.

Visibility in the Alert Monitor

The alert monitor offers several views which are graded according to monitoring/analysis requirement. You need to specify separately the visibility level of your monitoring object and each of its attributes.

The user of the alert monitor can set the visibility in order to screen out objects and attributes that he or she does not wish to see. However, no matter what visibility level is currently selected, a monitoring attribute and object that have an alert are ALWAYS visible. A system administrator or operator cannot miss an alert because you have set "AL_VISIBLE_DEVELOPER" and the user has selected the operator view (AL_VISIBLE_OPERATOR).

Recommendation: Select the visibility whose character best matches that of your object or attribute. If the object and attribute need to be monitored all the time, then they belong in the operator view (AL_VISIBLE_OPERATOR). If you need to be an SAP rocket scientist to interpret the information that is displayed, then they belong in the expert view (AL_VISIBLE_DEVELOPER).

Messages in the Alert Monitor

A monitoring object, attribute, or alert is represented directly in the monitor by the name that you give it in the CREATE_ATTACH call. For this reason, you should choose names that are likely to mean something to average mortals.

In addition to a name, however, you can also offer F1 help for an object, attribute, or alert. Each CREATE_ATTACH module lets you specify a standard R/3 message from table T100 that is to be associated with an object, attribute, or alert. With F1, the user can display the message and its documentation (F1 documentation class NA).

It is essential that you provide useful messages for describing monitoring objects and attributes and for explaining what to do if an alert occurs. You may well provide the user with excellent information and scintillating alerts. But if you do not bother to choose readable names or to provide useful explanations, the average user will only be irritated by your efforts.

Note that messages uploaded through the XMI interface (CCMS interface to external system management products) cannot be used as alert monitor messages.

Recommendation: Define and document new messages specially for each monitoring object, attribute and alert that you create. Specify them in your data supplier. Pre-existing messages are not likely to be suitable, because describing an object or attribute or explaining an alert are new uses for the T100 messages.

Properties: MTE Classes, Customizing Groups

MTE classes and customizing groups make it possible to manage MTE properties as global attributes of MTEs, applicable to all instances of a monitoring object or attribute.

Example: More than one instance of a monitoring object or attribute may exist at one time in the alert monitor. For example, the "CPU" monitoring object exists for each server that is represented in the alert monitor. Each CPU object is created with the same MTE class. By way of this shared MTE class, a user can manipulate the properties for all instances of a CPU object or attribute.

Further, the MTE class makes it possible to store sets of properties in alert monitor 'properties variants'. Properties variants allow users to define separate policies for monitoring the system and to transport these policies from one system to another. One variant may make CPU threshold settings for production systems. Another may make different settings for test systems.

The monitoring architecture also allows an MTE to have its own private version of an MTE class or customizing group. This makes it possible, for example, to change an alert threshold for a single instance of an MTE without affecting other instances of the MTE.

Properties include customizing settings of an object or attribute, and range from the descriptive message associated with an object or attribute to the alert thresholds that may be associated with a monitoring attribute. You can set most object and attribute properties from your data supplier. The user can also change properties with transaction RZ21.

MTE classes are also used in monitor definitions. The rule CCMS_GET_MTE_BY_CLASS allows users to select monitoring objects and attributes for a monitor by MTE class.

Here are the guidelines to follow for specifying MTE classes and customizing groups for your monitoring objects and attributes:

Setting up MTE classes:

- Each monitoring object and each monitoring attribute should have its own distinct MTE class.

  This is important to allow monitoring objects and attributes to have individual properties, to allow fine-grained selection of monitoring tree elements by MTE class, and to ensure that the SAP_DEFAULT properties variant can be correctly generated.

- Method assignments (alone among properties) are inherited. This means that

  - You should specify methods (as available) for every monitoring object in the SALI_MO_CREATE_ATTACH call:

    ```
    TOOL_COLLECTING = "My Data Supplier"
    TOOL_ONALERT    = "Object Auto-Reaction Method"
    TOOL_ANALYZE    = "Object Analyis Method"
    ```

- Specify methods for a monitoring attribute if the attribute needs different methods than its parent object.
- Omit method assignments – and their accompanying parameters – for a monitoring attribute if the methods of the parent object are also appropriate for the monitoring attribute.

  Example: Assume an object "Users Logged On." Assume further that the object has the attributes "Number of Users" and "User Logged on in Multiple Servers".

  In the "Users Logged On" object, you specify "SM04" in the TOOL_ANALYSIS parameter in the CREATE_ATTACH function module call. This transaction, the user overview transaction, is also appropriate as the analysis method for "Number of Users". In the CREATE_ATTACH call for this attribute, you could therefore omit the TOOL_ANALYSIS parameter. The "Number of Users" attribute would then inherit the SM04 method from its parent object, "Users Logged On."

  The "User Logged on in Multiple Servers" attribute, by contrast, needs a different analysis tool (transaction ST07). For this attribute, you would use the TOOL_ANALYSIS parameter to prevent the attribute from inheriting the analysis method from "Users Logged On."

Setting up customizing groups: In general, each monitoring attribute should have its own alert thresholds and therefore its own individual customizing group. You should therefore ensure that you use unique customizing group names when you create / attach attributes.

There is also an important **programming practice** to follow: You must fully specify the settings for your monitoring object and attributes in your data supplier, regardless of whether you expect an MTE class or customizing group to be already defined. The only exception: methods that should be inherited, as described above.

In the event that your data supplier is running for the first time (since a system start, for temporary objects), then your program must be able to provide a complete set of properties for the monitoring object(s) and attribute(s) that it creates.

Some Necessary Background Information...

When you create / attach a monitoring object or monitoring attribute, you specify up to three types of settings for the object or attribute. These are:

- General properties (T100 descriptive message, visibility, and so on)
- Method assignments (data collection, analysis, and auto-reaction methods to use)
- Thresholds (alert triggering, for monitoring attributes only).

For interactive users, these customizing functions are available from the alert monitor (transaction RZ20) or in the customizing transaction (RZ21).

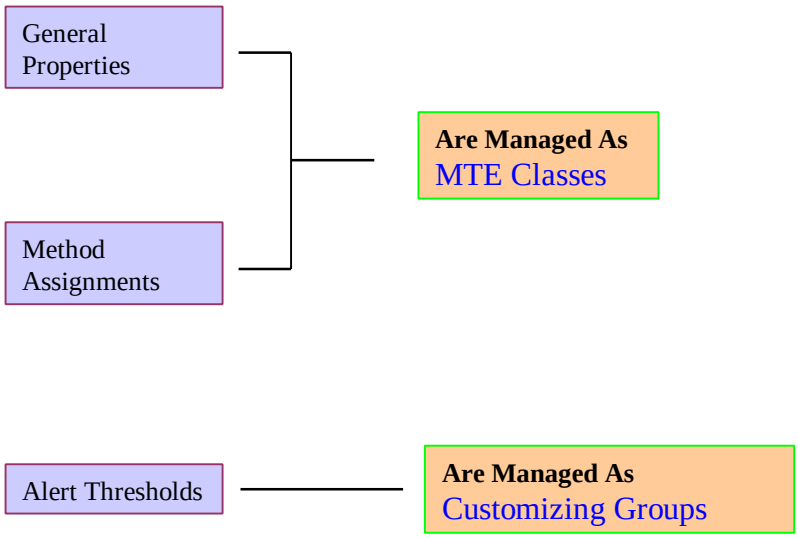

Figure 3. MTE Classes, Customizing Groups, and Method Assignments to MTE Classes.

The monitoring architecture manages general properties and method assignments as **MTE classes**.

Threshold customizing is stored by the monitoring architecture as **customizing groups**.

How are MTE classes and customizing groups managed in the monitoring architecture? According to a simple principle: the database always wins.

The diagram below shows you what you as a programmer need to know about creating and altering an MTE class or a customizing group.

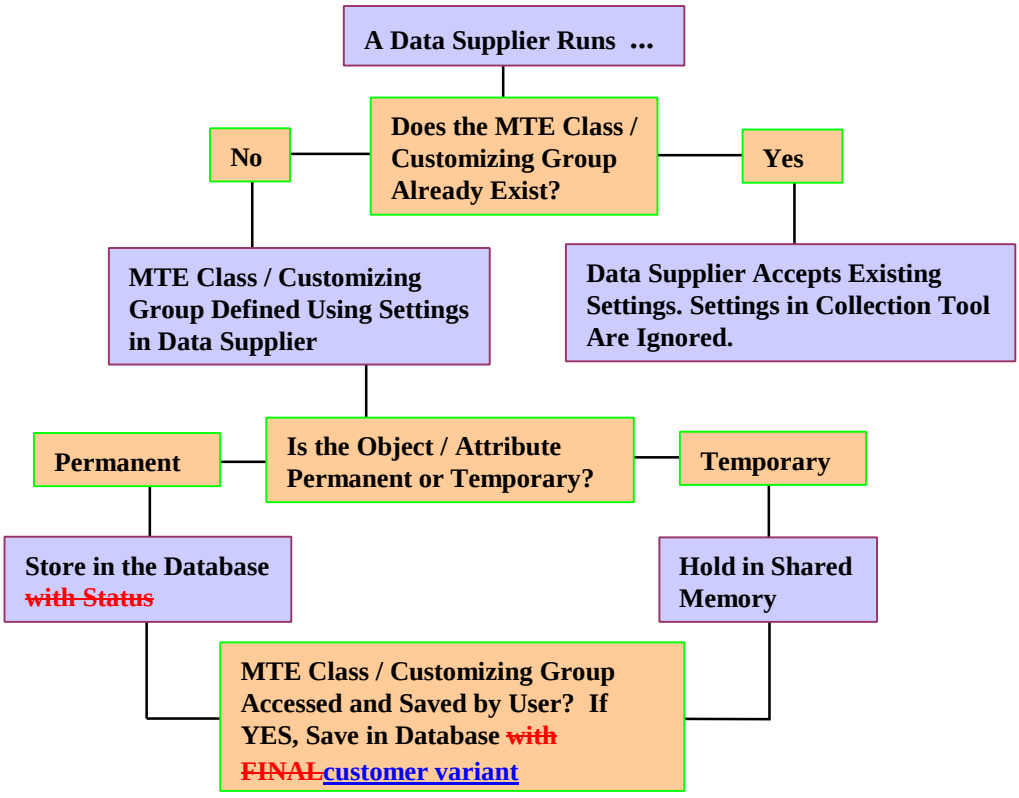

Figure 4. The Database Always Wins...

When a data supplier runs and calls a SALI_XXX_CREATE_ATTACH function module, the monitoring architecture checks to see whether the MTE class and/or customizing group specified in the call already exist.

If the data supplier is running for the first time, then the MTE class / customizing group do not already exist (unless a user has defined them in transaction RZ21). The monitoring architecture creates them using the specifications in the function module call. If the monitoring object or attribute is temporary (is deleted when an instance / the R/3 System is restarted), then the MTE class / customizing group is held only in shared memory. Otherwise, the MTE class / customizing group is written to the database.

All settings in permanent MTE classes and customizing groups are added automatically to the SAP_DEFAULT customizing variant.

The scenario "data supplier runs for the first time" can happen frequently with temporary objects and attributes. These MTEs are deleted, usually along with their MTE classes and customizing groups, when the R/3 instance in which they exist is restarted. When the instance is re-started, then the data suppliers run again and re-create their MTE classes and customizing groups.

If the MTE class or customizing group already exists, then the monitoring architecture ignores any settings made in the function module call. The data supplier attaches to the object or attribute using the MTE class / customizing group settings that have already been registered with the monitoring architecture.

To change settings, you must therefore do the following:

- In the case of temporary objects, delete the MTEs in question, together with their MTE classes, customizing groups, and method assignments; or
- In the case of permanent objects, use transaction RZ21 to refresh the SAP_DEFAULT properties variant. The refresh operation lets you load new settings for permanent MTEs into SAP_DEFAULT, which is the basis for all other properties variants.

Structuring Your Monitoring Tree: Summaries and References

You can, as described in the previous section, decide where your monitoring functionality will appear in the alert monitor. It can have its own system-wide context or can be inserted under a previously-existing MTE. For example, your monitor could appear under the MTE of each application server in an R/3 System.

Within your monitor, the monitoring architecture gives you a similarly free hand in structuring the information that you are presenting:

- With summary MTEs, you can group MTEs that have actual monitoring functionality (monitoring objects and attributes). Summaries let you insert purely organizational or structural elements in your monitoring tree.

  Example: Assume your monitor reports on two different aspects of your component. You could structure the monitor by grouping the objects and attributes that relate to each aspect under separate summary MTEs.

  Summaries can be stacked, with one summary containing others. However, summaries cannot be inserted under monitoring objects or monitoring attributes.

- References let you link existing monitoring functionality into your monitoring tree. You can, for example, pull related information into your monitoring tree without having to create a redundant monitoring object or attribute. You can also specify your own identification text for the referenced MTE.

Users have similar capabilities to structure **monitor definitions** by using virtual nodes to organize the MTEs selected by a monitoring definition and selection rules to include sets of MTEs by MTE class.

The advantage to you of doing your own structuring is that you can impose a permanent structure on your monitor. This structure is, for example, taken over by users if they select your monitoring tree by its root MTE or by any MTE above the summaries and/or references. For more information, see the online documentation on defining your own monitors.

Defining a Monitoring Object

Defining (creating) or attaching to a monitoring object is the first step in monitoring your component. You create or attach to a monitoring object with the function module SALI_MO_CREATE_ATTACH.

The monitoring object and monitoring attribute function modules both work the same way: If no corresponding object or attribute exists in the monitoring architecture, then it is created. Otherwise, your collection method "attaches" to the already-existent object or attribute.

You need to make these specifications when you create/attach to a monitoring object:

- **Name**: Choose a name for the monitoring object that will mean something to a user of the alert monitor. The name is displayed in the alert monitor. An internal technical name is therefore not a good idea. Spaces and punctuation are allowed in the name.
- **Position in the monitoring tree**: Here, you need to specify the "parent MTE" of your monitoring object – where your object will appear in the monitoring tree.

  Typically, you will have the name of an MTE that you know will always be available. The alert monitor provides variables that represent some of these MTEs: the current R/3 System, the current host, or the current R/3 instance. You may also create your own context. See *Creating Your Own Monitoring Context* on page 56.

  Once you have the name of an MTE, you can get its TID (its handle) by calling SALI_MT_GET_TID_BY_NAME in your data supplier. This TID is then the parent TID in your create/attach call for the monitoring object. You will find examples in the sample data suppliers in this document or online (RSDSSMPL*.)

  Note: You can create your own monitoring context in the alert monitor with the SALI_MC_CREATE_ATTACH function module. The resulting root node is inserted into the alert monitor at the same level as R/3 Systems. The function module returns a TID which you can use as the root for attaching your monitoring objects and attributes. See *Creating Your Own Monitoring Context* on page 56for more information.

  In the future, the monitoring architecture will also make it possible to offer a monitoring context in your own transaction. This will let you offer your users a monitoring transaction separate from the system monitoring transaction, RZ20.

Defining a Monitoring Attribute

The second step in monitoring your component is to create one or more monitoring attributes. A monitoring attribute is the MTE into which you report performance data or messages and in which the monitoring architecture generates alerts. You insert attributes in the monitoring tree under the monitoring object to which they pertain.

You should define a separate monitoring attribute for each type of information that you wish to monitor. For example, if you wish to monitor for object XYZ two performance values and also the occurrence of single messages, you would need to define three monitoring attributes. All three attributes will appear under the object in the alert monitor.

The type of information that you are reporting determines the function module you will need to use:

- SALI_PERF_CREATE_ATTACH to create a performance monitoring attribute. Use to report numeric performance values, such as percentages, sizes, throughput, and so forth into the monitoring architecture.
- SALI_SMES_CREATE_ATTACH for a status (single-message) attribute. Use to report discrete status messages into the monitoring architecture. The attribute displays only a single message at a time.

  Example: You could use the attribute to collect all error messages that a component might issue. See the sample program for suggestions on integrating the data supplier into your component.
- SALI_MSC_CREATE_ATTACH for a (message-container) log attribute. You can use this attribute to create a log or trace facility for your component. The monitoring architecture provides the logging/tracing functionality, you provide the messages.

You need to make these specifications when you create/attach to a monitoring attribute:

- **Name**: Choose a name for the monitoring attribute that will mean something to a user of the alert monitor. The name is displayed in the alert monitor. An internal technical name is therefore not a good idea.

  Performance attributes: You can make the name of a performance attribute more understandable by specifying a **Unit** to be displayed with the name. The unit specifies the unit of measure of the performance value. Together with the unit, you can also specify the number of decimal places to insert into the value, where sensible.
- **Position in the monitoring tree**: Here, you need to specify the TID that you obtained when you create/attached to the monitoring object.
- **Severity**: Here, you need to specify in the range from 0 (not important) to 255 (critical) how important the attribute is. This value is used, together with the alert criticality (red, yellow, green), to rank alerts across objects and attributes.

  Example: The number of users on an R/3 instance is not particularly critical in monitoring an R/3 System. The severity of the attribute is correspondingly low. The condition of the enqueue system, by contrast, is critical to the continued functioning of an R/3 System. Its severity in the alert monitor is correspondingly high.

  See also *Controlling Alert Ranking* on page 50
- **Visibility**: See *Visibility in the Alert Monitor* on page 41.

Controlling Whether and How Messages Trigger Alerts

Both status attributes (status messages or single messages) and log (message container) attributes allow you to control whether messages can lead to alerts, regardless of the criticality assigned to a message. You can allow messages to be reported and displayed in the alert monitor, but prevent them from triggering alerts.

- Status attributes offer an "alert mode" switch which lets you turn alert triggering on and off.

  Status attributes also offer the capability of "shifting" the criticality of an alert. For example, you can prevent a status attribute from triggering a red alert by "shifting" all red alerts to yellow alerts.

  Parameters are shown in the sample code below.

```
CALL FUNCTION 'SALI_SMES_CREATE_ATTACH' " Create status message attribute
  EXPORTING
    ...
    " Specify whether to trigger alerts for messages or whether
    " to limit the criticality of such alerts
    SMES_ALERT_MODE = AL_SMSG_ALMODE_ALWAYS
                    " Should the alert monitor ignore a message for purposes
                    " of triggering alerts?  Constants:
                    " AL_SMSG_ALMODE_ALWAYS:  Always trigger an alert if the
                    "   message criticality warrants an alert.
                    " AL_SMSG_ALMODE_VALUE_CHG:  Trigger an alert only if
                    "   the alert criticality changes and the message
                    "   criticality warrants an alert.  Alert triggered in any
                    "   case if no alert exists already.
                    " AL_SMSG_ALMODE_MSG_CHG:  Trigger an alert only if the
                    "   reported message, the criticality, or the severity
                    "   changes and the message criticality warrants an
                    "   alert.
                    " AL_SMSG_ALMODE_NEVER:  Never trigger an alert,
                    "   regardless of message criticality (ignore messages
                    "   for purposes of alerts).
    SMES_ALERT_SHIFT = AL_SMSG_ALSHIFT_UNCHG
                    " Should the alert monitor modify the criticality of any
                    " alerts triggered by messages?  Constants:
                    " AL_SMSG_ALSHIFT_UNCHG:  Do not modify alert
                    "   criticality.  Use the criticality specified in
                    "   SALI_SMES_REPORT_T100_MESSAGE.
                    " AL_SMSG_ALSHIFT_R_AS_Y:  Issue only yellow alerts.
                    "   Change red alerts to yellow alerts.
                    " AL_SMSG_ALSHIFT_Y_AS_R:  Issue only red alerts.
                    "   Change yellow alerts to red alerts.
                    " AL_SMSG_ALSHIFT_RAY_YAG:  Reduce criticality.  Change
                    "   red alerts to yellow, yellow alerts to green.
    SMES_CUSTOMIZING_GROUP = MY_CUSTOMIZING_GROUP
                    " Customizing group for changing these settings in the
                    " alert monitor with transaction RZ21.
```

- Message containers offer a filtering mechanism for messages: Each message may have its own numerical severity. This allows the alert monitor to do a preliminary filtering of messages according to the combination of severity and criticality before continuing with the normal alert triggering evaluation.

  This message severity is used only for filtering messages. Any alert triggered as a result of a message (after filtering) has the severity specified for the monitoring attribute.

  Parameters are shown in the sample code below.

```
CALL FUNCTION 'SALI_MSC_CREATE_ATTACH'  " Create log (message container)
                                        " attribute
...
    MSC_RAISE_VALUE = AL_VAL_GREEN      " Filter for criticality of messages
                                        " Messages with higher criticality
                                        " and severity higher than
                                        " MSC_RAISE_SEVERITY will be
                                        " considered for triggering alerts.
                                        " Works with MSC_RAISE_SEVERITY.
    MSC_RAISE_SEVERITY = 255            " Filter for severity of messages
                                        " Messages with higher severity are
                                        " considered for triggering alerts
                                        " Works with MSC_RAISE_VALUE.  Here,
                                        " the minimum is set to criticality
                                        " green and severity 255.  This
                                        " excludes all green messages,
                                        " allows any yellow or red msg to
                                        " generate an alert.  YELLOW and
                                        " 100 would screen out GREEN msgs
                                        " and YELLOW messages with a
                                        " severity of less than 100.
    MSC_CUSTOMIZING_GROUP = MY_MSC_GRP  " Customizing group for changing
                                        " these settings in the alert
                                        " monitor with transaction RZ21.
```

Note: See also *Controlling Alert Ranking* on page 50for related information on specifying alert severity and criticality.

Background Information: Alert Screening

Here is how the alert monitor screens messages that have been added to a log (message container) monitoring attribute.

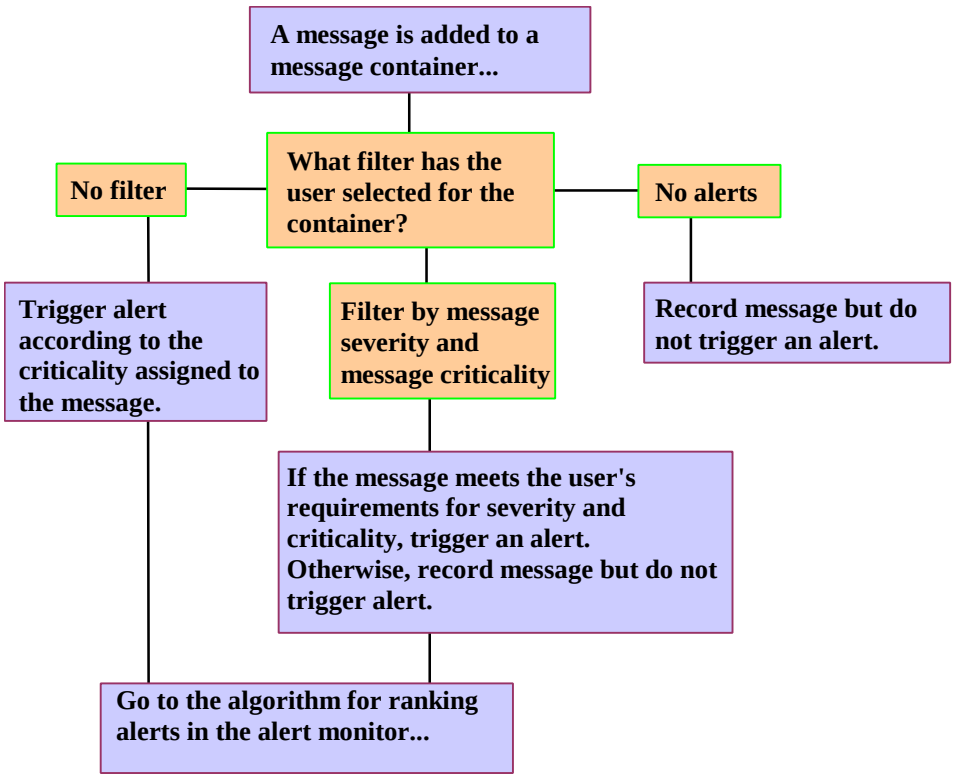

Figure 5. Filtering Messages in a Message Container Before Triggering Alerts.

Here is how the alert monitor screens messages that have been reported to a status attribute (single-message attribute).

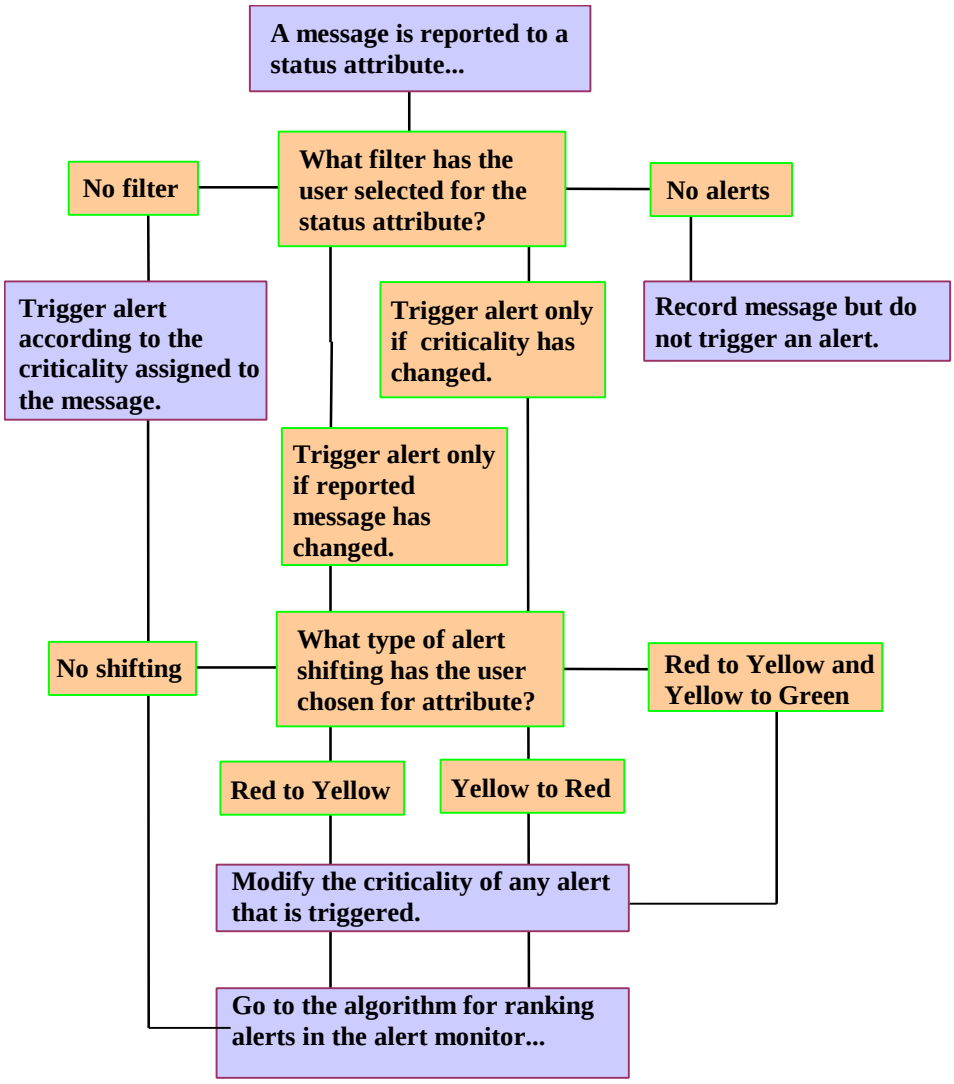

Figure 6. Suppressing or Modifying Alerts in a Status Attribute.

Controlling Alert Ranking

The alert monitor is able to rank alerts and pass the most important of a set of alerts up the monitoring tree. This section explains how to control this ranking of alerts.

Example: If the most important problem in the monitor is currently a yellow alert, then the yellow warning coloring and the yellow alert text are passed up the monitoring tree from the monitoring attribute where the alert was generated. Even if the monitoring tree is closed up to the root monitoring tree element, you can still see that the most serious problem in the system is a yellow alert, and (if desired) you can still display the yellow alert text.

> Note: For status attributes (single messages) and log attributes (logs and traces), you can also control whether and how the alert monitor triggers alerts. Please see *Controlling Whether and How Messages Trigger Alerts* on page 48 for more information.

The alert monitor uses two measures to specify how important an alert is and how to rank alerts:

- **Severity**. The numerical severity (from 0 least severe to 255 most severe) expresses the importance of an MTE (a monitoring attribute).

  Parameter: MTE_SEVERITY in the CREATE_ATTACH function modules

- **Alert criticality:** The assignment of the state of a monitoring attribute to the category green (normal, OK) or to one of the alert categories (yellow (warning) or red (problem / error)). The criticality depends upon the alert thresholds that you specify for performance attributes or the criticalities that you associate with messages for status and log attributes.

  Parameters:

  - Performance attributes: Alert thresholds in the SALI_PERF_CREATE_ATTACH function module

  - Status and log (message container) attributes: Criticality assignment (Red, Yellow, Green) with the MSGVALUE parameter in the SALI_SMES_REPORT_T100_MESSAGE function module (status messages); VALUE and SEVERITY parameters in the SALI_MSC_REPORT_T100_MESSAGE function module (logs).

Guidelines for Controlling Alert Ranking

- **Use alert criticality to express the seriousness of a condition with respect to a component.**

  A yellow alert signals a warning; a red alert signals an error or failure.

  Be aware that a red alert always is ranked higher than a yellow alert, regardless of the severity (importance) associated with a monitoring attribute. The logic: an outright error or failure should always take precedence over a warning, even if the warning pertains to a more important component.

  Here is sample coding for alert criticality. For customizing, alert criticality is a property of the customizing group of a monitoring attribute. Please also see *Controlling Whether and How Messages Trigger Alerts* on page 48.

```
CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'  " Create performance attribute
  EXPORTING
    ...
    " Set alert criticality by setting alert thresholds...
    PERF_THRESHOLD_GREEN_TO_YELLOW = 50  " Trigger yellow alert
    PERF_THRESHOLD_YELLOW_TO_RED   = 100 " Trigger red alert
    PERF_THRESHOLD_YELLOW_TO_GREEN = 40  " Return condition to
                                         " green from yellow
    PERF_THRESHOLD_RED_TO_YELLOW   = 90  " Return condition from
                                         " red to yellow. Return
                                         " values lower to prevent
                                         " 'flickering'.
    PERF_CUSTOMIZING_GROUP = "perf_cust_group"
                                         " Customizing group for changing
                                         " criticality settings with TX
                                         " RZ21.

CALL FUNCTION 'SALI_SMES_REPORT_T100_MESSAGE'
              " Report a message into an existing status attribute
  EXPORTING
    ...
    MSGVALUE = AL_VAL_GREEN  " Specify the alert criticality of the
```

```
                              " reported message.  RSALEXTI constants:
                              " AL_VAL_GREEN:  Situation normal:  no alert.
                              "   Success message and the like.
                              " AL_VAL_YELLOW:  Interpret message as warning
                              "   and trigger yellow alert.
                              " AL_VAL_RED:  Interpret message as failure and
                              "   trigger red alert.
   " Customizing group does not apply to message criticalities.  See however
   " Controlling Whether and How Messages Trigger Alerts on page 48

CALL FUNCTION 'SALI_MSC_REPORT_T100_MESSAGE'
              " Report a message into an existing log (message container)
              " attribute
  EXPORTING
    ...
    VALUE = AL_VAL_GREEN      " Specify the alert criticality of the
                              " reported message.  RSALEXTI constants:
                              " AL_VAL_GREEN:  Situation normal:  no alert.
                              "   Success message and the like.
                              " AL_VAL_YELLOW:  Interpret message as warning
                              "   and trigger yellow alert.
                              " AL_VAL_RED:  Interpret message as failure and
                              "   trigger red alert.
   SEVERITY = 50              " Message severity (0 (unimportant) - 255 (very
                              " important))
                              " Specify the severity of the reported message.
                              " This severity is used ONLY for screening the
                              " message before proceeding to evaluation for
                              " alert triggering.
   " Customizing group does not apply to message valuation.  See however
   " Controlling Whether and How Messages Trigger Alerts on page 48
```

- **Use severity to express the importance of the component associated with an alert. Severity is used to sort alerts that have the same criticality.**

  Example: Assume you are writing a data supplier for a critically important R/3 component. You should give the monitoring attributes that you define for this component a very high numerical severity. The high severity expresses the importance of the component.

  Less important components should have lower severity values.

  The result: a red alert from the more important component will be ranked ahead of a red alert from less important components. The alert monitor will highlight the more important red alert in its display.

  There are two main schemes for assigning severities:

  - The normal case: All monitoring attributes of a monitoring object share the same severity.

    Uniform severities are for example right when the significance of alerts derives primarily from the component that is being monitored. Assigning uniform severities then ensures that all alerts pertaining to the component have the severity (importance) assigned to the component.

    Example: Assume a monitoring object "Users" that has the attributes "number of users" and "user has reserved work process (debugging...)". The main importance for system monitoring attaches to the component "users"; the monitored conditions represented by the monitoring attributes do not have an independent importance. The object and attributes should therefore have the same severity.

    ```
    CALL FUNCTION 'SALI_MO_CREATE_ATTACH'  " Create monitoring object
    ```

```
  EXPORTING
    ...
    " No severity can be defined for objects -- only for attributes
    ...

CALL FUNCTION 'SALI_PERF_CREATE_ATTACH' " Create performance attribute
  EXPORTING
    ...
    MTE_SEVERITY = 50                   " Range 0 (unimportant) - 255
                                        " (critical).  Same severity for
                                        " all attributes of an object.
                                        " Severity is later customizable
                                        " as a property of the MTE class

CALL FUNCTION 'SALI_SMES_CREATE_ATTACH' " Create status message attribute
  EXPORTING
    ...
    MTE_SEVERITY = 50                   " Uniform severity for attributes
                                        " of this object

CALL FUNCTION 'SALI_MSC_CREATE_ATTACH'  " Create log (message container)
                                        " attribute
  EXPORTING
    ...
    MTE_SEVERITY = 50                   " Uniform severity for attributes
                                        " of this object
```

- Monitoring attributes have importance independently of one another and their parent monitoring object (component).

  In this unusual case, assign separate severities to each monitoring attribute according to the significance that each has. Any alerts will be ranked according to the severity specified for their monitoring attributes in the CREATE_ATTACH function module calls.

  Example: The standard system log monitoring object in the alert monitor is the parent for a wide variety of component-oriented system log message containers. These individual log (message container) attributes have widely differing importance. Further, the importance of each attribute has little direct connection to the importance of the system log component as such.

  In this case, it might be desirable to assign a very high severity to the database log attribute in the system log, since it may report system-critical problems. The "miscellaneous" log attribute, by contrast, could have a much lower severity, since the system log messages reported to it are likely to be much less important than database errors.

  The coding might then look like this:

```
CALL FUNCTION 'SALI_PERF_CREATE_ATTACH' " Create performance attribute
  EXPORTING
    ...
    MTE_SEVERITY = 200                  " Range 0 (unimportant) - 255
                                        " (critical).  An important
                                        " performance attribute.  Later
                                        " customizable as a property of
                                        " attribute's MTE class
    MTE_CLASS = CRITICAL_PERF_CLASS     " Special MTE class for this
                                        " attribute

CALL FUNCTION 'SALI_SMES_CREATE_ATTACH' " Create status message attribute
  EXPORTING
    ...
    MTE_SEVERITY = 100                  " Not so important status message
```

```
                                          " attribute
      MTE_CLASS = NOT_SO_CRITICAL_CLASS   " Special MTE class for this
                                          " attribute
  ...
```

- **Severity scale:** The default severity in the alert monitor is 50. The table below shows a possible scale for rating severity. Note, however, that there is no official scale for ranking severities.

Severity Value	**Meaning**
10 - 100	An alert from this attribute signals a condition which may reduce performance. Neither the integrity of data nor the availability of the system or its components is endangered. Default: 50 Example: Too many users on a server.
100-200	An alert from this attribute signals a condition that may result in unavailability of a system component or function. Neither the integrity of data nor the availability of the system are immediately threatened. Example: The spool service is not working correctly (print output may be delayed or interrupted).
200-255	An alert from this attribute signals a condition that may immediately threaten the availability of the system and/or the integrity of data.

Background Information: Alert Ranking Algorithm

The basic rules in the algorithm are as follows:

- Alert ranking takes place recursively from individual monitoring attributes upward in the monitoring tree. At each level of the tree, the most important alert in that portion of the tree is displayed.
- Alert criticality takes precedence over severity.

  A red alert is always ranked higher than a yellow alert, even if the "yellow" monitoring attribute has a higher severity (is more important). The reason for this rule: A yellow alert is only a warning. A red alert, by contrast, signals that an actual error or problem has occurred.
- At the same level of alert criticality, higher severity takes precedence.

  If all of a set of alerts is either red or yellow, then the alert from the monitoring attribute that has the highest severity is ranked highest. A red alert for "update errors" with a severity of 100 takes precedence over a red alert for "too many users" with a severity of 50.

The illustration below shows how the ranking algorithm in the alert monitor works (for all alerts).

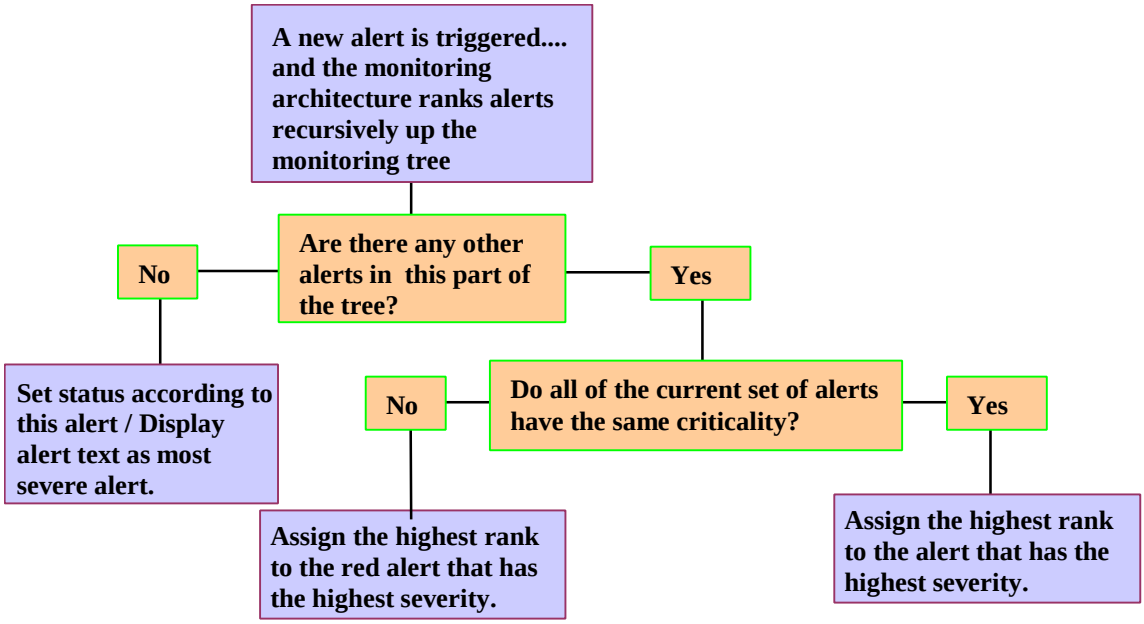

Figure 7. Ranking Alerts.

Setting Alert Thresholds for a Performance Monitoring Attribute

When you create a performance monitoring attribute, you need to tell the alert monitor when it should generate alerts for the attribute and how to manage these alerts.

You need to specify the following:

- **Default threshold values:** Your data supplier must specify the levels at which yellow and red alerts are to be generated.

  These levels are also used in the "current condition" display in the alert monitor. You therefore also need to set the thresholds at which the monitor switches from red back to yellow, or yellow back to green.

- **Should alerts be ignored during a warm-up time?** Activating this option is only sensible for monitoring objects such as buffers. A buffer must be filled before it is useful to monitor its accuracy.

- **How to evaluate the data that is reported**.

  For a performance attribute, you need to decide whether to evaluate performance values (a response time, for example), or the frequency per minute with which values are reported (the frequency with which a message is issued, for example).

  Then you need to decide whether to trigger alerts when your thresholds are exceeded or when they are not met.

- **How to manage alerts.**

  At this time, the alert monitor does not guarantee that all alerts will be kept. It is possible (though unlikely) for alerts to leave the monitor without having been seen by the operator. In the future, the alert monitor will guarantee that no alerts can be lost.

For the present, however, you need to specify **how many alerts to keep**. If more alerts are triggered and this limit is exceeded, then some alerts will be discarded.

You can also specify the **discard order** for alerts when the limit is exceeded. Sensible discard orders are for example discarding the oldest alerts (this is the default, FIFO order) or discarding the least severe alerts.

- **How long reported data remains valid.**

  You can specify how long the monitoring architecture should regard data as valid. Should data not be refreshed within this interval, then the monitoring architecture colors the affected MTE white and displays the message 'value is old' in the alert monitor display. As a rule of thumb, you should set the aging limit to a little bit more than twice the length of the data collection refresh interval. That allows the monitoring architecture to try to start passive collection methods twice before the aging limit runs out.

  You can turn off aging by setting the parameter, MTE_SECONDS_UNTIL_SET_INACTIVE, to 0. This is often sensible for active data suppliers. The monitoring architecture has no control over how and when active data suppliers are started. The monitoring architecture can therefore not guarantee that an active data supplier will be started within the aging limit.

Creating Your Own Monitoring Context

Create your own context if you wish to have your own major, system-global branch of the monitoring tree in the alert monitor. In the alert monitor, a context appears directly under the name of an R/3 System. You could, for example, create a monitoring context for the component that you are adding to the alert monitor.

In general, you should create your own context only if you are collecting global system data (that is, data that is not separately generated in each application server). Otherwise, you should insert your monitoring objects and attributes under a standard context, such as the application server context.

Create or attach a monitoring context with SALI_MC_CREATE_ATTACH. The function module returns a TID that you can use as the parent for your monitoring objects (SALI_MO_CREATE_ATTACH). Please see the online help in the *Function Builder* for more information on using this function module.

In an MTE name, the context is always the second element of the name. Example: "Spool" is the context of the "MaxWaitTime" monitoring attribute, as shown in the full name of the monitoring attribute: \BCE\Spool\...\SystemWide\MaxWaitTime.

System-Wide Contexts

A context may appear in each application server (more exactly, in each monitoring segment) or may be 'system wide'. A system-wide context is designated as such in the CREATE_ATTACH call and may appear only once in an R/3 System.

Note: The type of context that you create should reflect an efficient data collection and storage scheme. Do not try to collect server-specific data in a single system-wide context; you cannot issue RFC calls from a passive collection method, and you will also have to manage for yourself such issues as which servers are active and so on. Rather, you should collect and store server-specific data in a server-specific context, system-global information in a system-global context. You can use monitor definitions in the alert monitor to deliver to your users the view of your data that you wish them to have, regardless of the structure of your data collection and storage.

Client-Specific Contexts

As of Release 4.6A, you can define client-specific contexts. For example, you could define contexts for Component A for clients 100, 200, and 300. The client assignment is held as a property of a context in the ALCONSEG table. This property is not currently used in the monitoring architecture (for example, in selecting data to display). However, in the future, the client assignment property will make it possible to build client-specific monitors.

Discarding a Monitoring Object and Attributes

You can destroy a monitoring object or attribute when it is no longer needed. There are two ways to do this. You can either:

- Allow objects and attributes to be deleted when an R/3 instance (or the entire system) is restarted. This practice could be called "implicit discarding" of MTEs.

  You accomplish this by allowing the monitoring architecture to assign temporary number range numbers for the objects and attributes. This practice makes the objects and attributes into temporary objects, from the perspective of the monitoring architecture. For more information, see*Number Range Management* on page 39.

- Explicitly destroy an object or attribute with the function module SALI_MT_MARK_FOR_DESTRUCTION.

  The monitoring architecture deletes the MTE as soon as all alerts that pertain to it have been completed.

  Note, however, that the MTE class and customizing group (if applicable) of the MTE persist after the MTE is deleted. This means the following: when the data supplier next runs (in the same instance), it must accept the MTE class and customizing group settings that were already registered with the monitoring architecture. Usually, this is not a problem. Either the settings match the values in the data supplier. Or the settings have been edited by a user and therefore in any case have precedence over the settings in the data supplier.

Example (implicit discarding): the spool system creates a monitoring attribute for every R/3 printer with which a problem is reported. These attributes are created dynamically, as needed. It is of course not sensible to create attributes for 2,000 printers at system start, especially since most attributes will never be needed.

The spool system creates "printer" attributes as temporary MTEs. They accumulate slowly as long as an R/3 instance continues to run. Then they, their associated MTE classes and customizing groups, and any incomplete alerts are deleted when the instance or system is restarted.

During development (and only during development), you can discard a permanent MTE (one that uses permanent numbering (auto numbering and externally-assigned numbering)) and its subtree by MTE class using SALH_MTE_DESTROY_BY_CLASS. This function module makes a clean sweep of the MTE and its properties so that you can, for example, make a change in the name of an MTE in the data supplier program. You can achieve the same effect by using temporary MTEs, which are entirely deleted when the application server on which they exist is restarted.

Implementation Techniques and Tips

Preparing a Function Module for Use as an Alert Monitor Method

The alert monitor can directly call a function module for use as a data collection, analysis, or auto-reaction method. The advantage: the alert monitor can provide the function module with the current TID (MTE identification) or AID (alert identification), and can also pass other parameters to the function module.

The prerequisite: the function module must meet the interface requirements of the monitoring architecture. To create a function module that meets these requirements, do the following:

1. Copy the pre-defined function module SALT_TOOL_REFERENCE_MODULE.
2. Add your functionality to the function module. Extend the interface with additional parameters, if required. The parameter values should be accessible to the alert monitor (for example, as SY-... system fields).

   Do not delete any of the original parameters, as these are needed by the monitoring architecture.
3. Use transaction RZ21 to define the function module as a method. Specify any parameters that you have added in the method definition. See also *Defining Methods in Transaction RZ21* on page 37.

Auto-Reaction Methods

Prior to Release 4.6A, it was not possible to send mail from a method by way of SAPoffice. Instead, a method needed to trigger an external mail program.

As of Release 4.6A, mailing with SAPoffice is supported. A sample auto-reaction method is defined as method CCMS_OnAlert_Email, calling function module SALO_EMAIL_IN_CASE_OF_ALERT. This function module requires that the method definition specify a sender, recipient, and recipient type in the method parameters. These are part of the method definition, and are static within a method definition. (That is, to route mail to different recipients, for example, you will need to define separate methods.)

This function module reads the alerts for an MTE, packs them into an e-mail, and sends the e-mail to the recipient (an R/3 user) specified in the method definition.

Issuing Self-Monitoring Messages

The monitoring architecture is itself instrumented. Data on the monitoring architecture and on methods is displayed in the self-monitoring tree. This tree is available in the standard CCMS monitoring set SAP CCMS Technical Expert Monitors.

Data suppliers typically write their messages into a 'data supplier log' message container. You can, for example, write standard start and successful end messages in the log, as well as error messages.

Here is sample code:

```
INCLUDE RSALSMI.
INCLUDE RSALSMCI.
INCLUDE RSALSMFI.
```

```
* Local data
data self_moni_tid like alglobtid.

* self moni    send rt 39 ("method started....")
  PERFORM send_selfmon_msc_message
      USING  self_moni_tid     039  mt_tool_info-toolname
                                    space  mt_tool_info-toolname.

Your code...

* Method successfully ended...
  PERFORM send_selfmon_msc_message
      USING  self_moni_tid     101
             mt_tool_info-toolname space  mt_tool_info-toolname.
```

These routines are limited in functionality. For example, they assume that your program uses message ID RT (alert monitor interface). And only certain specified messages are reported with an alert criticality.

Should you need additional functionality, you can copy these routines to your own function group.

Sample Code: Performance Collection Method

```
*&---------------------------------------------------------------------*
*& Report RSDSSMPL_PERFORMANCE:  Template for a "data supplier" for    *
*&                 monitoring a performance parameter for any         *
*&                 R/3 component in the Release 4 Alert Monitor.      *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*
REPORT   RSDSSMPL_PERFORMANCE.

*---------------------------------------------------------------------*

* How the report works:  A quick summary.
* The report sees to it that your component is represented in the
* alert monitor.  It creates a monitoring object and monitoring
* attribute in the alert monitor.  The monitoring object represents
* your component.  The monitoring attribute represents the performance
* values that you are going to report for your component.  Each
* time the report runs, it creates the object and attribute if they
* are not yet available, or attaches to them, if they already have
* been created.  The report then passes the latest performance values
* from your component into the monitoring attribute.  The alert
* monitor then displays your data, triggers an alert if necessary,
* and so on.

* Embedding the report in your application: Typically, you can have
* the alert monitor take over the periodic execution of this report.
* Your application must provide a function with which this report
* can obtain the performance data that it is to report into the
* alert monitor. If such a function is already available, then no
* change is needed to your code.  You need only write a data supplier
* like this one, which accesses your data by way of the existing
* function.

*---------------------------------------------------------------------*
* Constants for the Release 4 Monitoring Architecture.
INCLUDE RSALEXTI.

* Constants for use in this sample program.
CONSTANTS:
* Monitoring object constants...
           SAMPLE_OBJECT_NAME   VALUE 'DemoObject: Number of Users'
              LIKE ALMTCUSGEN-MTNAMESHRT,
              " Name of the monitoring object to create and access in
              " the alert monitor. Enter the name that you wish to
              " display to users in the alert monitor.
              "
              " This sample report creates a new monitoring object
              " under the context "<System ID>/<Instance name>". The
              " object appears in the alert monitoring tree at the same
              " level as such static MTEs (Monitoring Tree Elements,
              " a generic term for alert monitor nodes) such as
              " "OperatingSystem" or "R3Services". The first run of the
              " program creates the object. Subsequent runs "attach" to
              " the newly-created object.

           SAMPLE_OBJECT_CLASS  VALUE 'SampleMTObjectClass'
              LIKE ALGRPCUSGE-CUSGRPNAME,
              " MTE class for the monitoring object: Enter a freely-
              " definable class name for certain customizing attributes
```

```
              " of your object.
              "
              " The MTE class lets you group the General Properties
              " Tool Specifications (TX RZ21) settings for an object.
              " Use: Lets all instances of an object share by reference
              " the same general properties and tool assignments.

* Monitoring attribute constants...
           SAMPLE_ATTRIBUTE_NAME    VALUE 'ExampleValue(NumberOfUsers)'
              LIKE ALMTCUSGEN-MTNAMESHRT,
              " Name of the monitoring attribute to be created for the
              " object above in sample_object_name. Enter the name that
              " you wish to display in the alert monitor. You report
              " information on an object only through the object's
              " monitoring attributes.
              "
              " This sample data supplier creates a performance
              " attribute for reporting the number of users logged on.

           SAMPLE_ATTRIBUTE_CLASS   VALUE  'SampleMTAttribClass'
              LIKE ALGRPCUSGE-CUSGRPNAME,
              " MTE class for the monitoring attribute: Enter a freely-
              " definable class name for the general properties and
              " method assignments of your MA.

           SAMPLE_CUSTOMIZING_GROUP VALUE 'SamplePerformanceGroup'
              LIKE ALGRPCUSGE-CUSGRPNAME.
              " Customizing group for the monitoring attribute.  Enter
              " a freely-definable name for the alert thresholds of
              " your monitoring attribute.
              "
              " A customizing group is an attribute-only complement
              " to the MTE class. The customizing group lets you store
              " attribute-specific customizing settings: the alert
              " thresholds for the monitoring attribute (TX RZ21).
              " Monitoring attributes of the same type, such as
              " number of users or response time, can share by
              " reference common threshold settings.

*--  global variables of program --*
DATA: SAMPLE_PERF_TID LIKE ALGLOBTID,
      " This is the TID (handle) in the alert monitor of the
      " monitoring object that this report will create and use.

      SAMPLE_PERF_VALUE TYPE I.
      " This is the receptacle for the performance value that this
      " program reports to the alert monitor.

*--  Start of the main program --*
START-OF-SELECTION.

* Step 1: Create/attach monitoring object and monitoring attribute.
* The first program run creates the object and attribute (makes them
* visible in the alert monitor). Subsequent program runs simply
* attach to the object, since it has already been created. Each
* program run also reports a value for the performance attribute.
  PERFORM CREATE_MO_MA_EXAMPLE CHANGING SAMPLE_PERF_TID.

* Step 2: Get the value to report in the alert monitor.
  PERFORM GET_SAMPLE_VALUE USING SAMPLE_PERF_VALUE.

* Step 3: Report the value in the alert monitor.
  PERFORM REPORT_SAMPLE_VALUE USING SAMPLE_PERF_TID SAMPLE_PERF_VALUE.

********************** end of the main program
```

```
* Collect performance data to report in the alert monitor. Here,
* we collect the number of users logged on. For your own data
* supplier, you would insert here a function module or program that
* collects the data that you wish to report.
FORM  GET_SAMPLE_VALUE USING SAMPLE_VALUE TYPE I.

  DATA: BEGIN OF USER_TBL OCCURS 100.
          INCLUDE STRUCTURE UINFO.
  DATA: END OF USER_TBL.

  CALL FUNCTION 'TH_USER_LIST'
       TABLES
            LIST = USER_TBL
       EXCEPTIONS
            OTHERS = 1.
  DESCRIBE TABLE USER_TBL LINES SAMPLE_VALUE.

ENDFORM.

*---------------------------------------------*
*-- create_mo_ma_example     ----------------------*
*---------------------------------------------*
*
* Create/attach to the monitoring object and monitoring attribute
*
FORM CREATE_MO_MA_EXAMPLE USING MA_P_TID LIKE ALGLOBTID.

  DATA: PARENT_TID LIKE ALGLOBTID,
        " This is the TID (handle) of the element in the monitoring
        " tree under which our new monitoring object and attribute
        " are to appear. Here, the parent is the current server.
        MO_TID     LIKE ALGLOBTID.
        " This is the TID of the new monitoring object created and
        " accessed by this program.

* Step 1: Get the TID of the parent element in the alert monitor tree
  call function 'SALI_MT_GET_TID_BY_NAME'
       exporting
            MT_FULL_NAME          = '\&SY\&INSTANCE_NAME'
                                    " Supported variables:
                                    " &SY: R/3 System name - C11
                                    " &N: SAP System number - 53
                                    " &HOSTNAM: Host name as in system
                                    "    profile parameter saplocalhost-
                                    "    host1
                                    " &INSTANCE_NAME: Instance name -
                                    "    profile parameter rdisp/myname-
                                    "    host1_C11_53
       importing
            TID                   = PARENT_TID
       exceptions
            unable_to_expand_name = 1
            name_not_found        = 2
            communication_failure = 3
            other_problem         = 4
            others                = 5.


* Step 2: Create a new monitoring object / attach to the existing
* monitoring object under parent_tid.
  CALL FUNCTION 'SALI_MO_CREATE_ATTACH'
       EXPORTING
            PARENT_TID            = PARENT_TID
            MO_NAME               = SAMPLE_OBJECT_NAME
```

```
            NUMRANGE              = AL_NR_TEMP " temporary number range
                                    " Sample uses the temporary number
                                    " range for the MO handle. Benefit:
                                    " the monitoring architecture
                                    " manages the number range; you
                                    " don't have to. See documentation
                                    " on number ranges.
            UNIQUENUM             = 0
                                    " ID number. Set to 0 if using
                                    " numrange = al_nr_temp.
            VISIBLE_ON_USERLEV    = AL_VISIBLE_OPERATOR
                                    " Specify level for display in
                                    " alert monitor:
                                    " al_visible_operator: Always
                                    "   visible.
                                    " al_visible_expert: Only visible
                                    "   in higher Analysis view
                                    " al_visible_developer: Only
                                    "   visible in Expert Analysis view
                                    " Note: Alerts are always visible.
            DESC_TEXT_MSGID       = 'RT'
            DESC_TEXT_MSGNO       = 092
                                    " T100-Message: Display as descript.
                                    " and F1-help anchor for MO (SE91)
            MT_CLASS              = SAMPLE_OBJECT_CLASS
       IMPORTING
            NEW_TID               = MO_TID
       EXCEPTIONS
            INVALID_TID           = 1
            UNABLE_TO_EXPAND_NAME = 2
            INVALID_PARAMETERS    = 3
            COMMUNICATION_FAILURE = 4
            OTHER_PROBLEM         = 5
            WRONG_SEGMENT         = 6
            INTERNAL_FAILURE_SALS = 7
            OTHERS                = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.


* Step 3: Create the performance monitoring attribute (MA)
  CALL FUNCTION 'SALI_PERF_CREATE_ATTACH'
       EXPORTING
            PARENT_TID                    = MO_TID
            MTE_NAME                      = SAMPLE_ATTRIBUTE_NAME
            MTE_CLASS                     = SAMPLE_ATTRIBUTE_CLASS
            MTE_NUMRANGE                  = AL_NR_TEMP " See above.
            MTE_UNIQUENUM                 = 0          " See above.
            MTE_SEVERITY                  = 50
                                            " Value from 0 (minor) to
                                            " 255 (severe) that shows
                                            " how important an alert in
                                            " the MA is. Ex: Too many
                                            " users is not important.
                                            " Failed updates are.
                                            " See documentation on
                                            " ranking alerts.
            MTE_VISIBLE_ON_USERLEV        = AL_VISIBLE_OPERATOR
                                                    " See above.
            MTE_KEEPALTYPE                = AL_KEEP_ALL
                                            " Keep alerts? Default:
                                            " keep all to limit in
                                            " mte_keepalmax.  Others:
```

```
                                  " al_keep_oldest
                                  " al_keep_newest
                                  " al_keep_highest
                                  " al_keep_slave
MTE_KEEPALMAX                   = 10
                                  " Keep a maximum of 10
                                  " alerts. Discard excess
                                  " according to keepaltype.
MTE_SECONDS_TIL_COLLECTINGTOOL = 240
                                  " How often should alert
                                  " start data supplier?
                                  " Here, the tool  will be
                                  " started approximately
                                  " every 240 seconds.
                                  " Set to 0 if not needed.
MTE_SECONDS_UNTIL_SET_INACTIVE = 900
                                  " Is element active?
                                  " If no value is reported
                                  " for 900 seconds, the
                                  " MTE in the monitor goes
                                  " white: Inactive. Set to
                                  " 0 to switch off.
MTE_SECONDS_WARMUPTIME          = 0
                                  " Ignore alerts for this
                                  " time period (sec.) during
                                  " warm-up.
MTE_F1_HELP_TEXT_MSGID          = 'RT'
MTE_F1_HELP_TEXT_MSGNO          = 093
                                  " T100-Message to display
                                  " as descript. and F1-help
                                  " anchor for MA (SE91).
PERF_CUSTOMIZING_GROUP          = SAMPLE_CUSTOMIZING_GROUP
PERF_SUBTYPE                    = AL_STD_NO_SUBCLASS
                                  " Monitor behavior: check
                                  " values of reported data
                                  " Alternate:
                                  " al_std_perf_freq_1_minute
                                  " check frequency of msgs.
PERF_RELEVANT_VALUE             = AL_PERF_RV_LAST
                                  " Evaluate most recent data
                                  " to trigger alert.
PERF_THRESHOLD_DIRECTION        = AL_THRESHDIR_ABOVE
                                  " Trigger alert when a
                                  " threshold is exceeded.
                                  " _below, when threshold
                                  " is not reached.
PERF_THRESHOLD_GREEN_TO_YELLOW = 50
                                  " Trigger yellow alert
PERF_THRESHOLD_YELLOW_TO_RED    = 100
                                  " Trigger red alert
PERF_THRESHOLD_YELLOW_TO_GREEN = 40
                                  " Return condition to
                                  " green from yellow
PERF_THRESHOLD_RED_TO_YELLOW    = 90
                                  " Return condition from
                                  " red to yellow. Return
                                  " values lower to prevent
                                  " 'flickering'.
PERF_UNIT_TO_DISPLAY            = 'Usrs'
                                  " Unit of performance attr.
                                  " for display in monitor.
                                  " Up to 4 characters.
PERF_DECIMALS                   = 0
                                  " Decimal places to display.
```

```
                                        " Functions as a conversion factor
                                        " for values, shifting the value the
                                        " number of decimal places you
                                        " specify. Ex.  If you specify 3,
                                        " then the value 2700 ms is displayed
                                        " as 2.7 (seconds).  The conversion
                                        " only the display.  Thresholds should
                                        " be entered in the unit in which
                                        " values are reported, in this example
                                        " in milliseconds.
            PERF_ALERT_TEXT_MSGID       = 'RT'
            PERF_ALERT_TEXT_MSGNO       = 001
                                        " T100-Message to display
                                        " on alert and as F1-help
                                        " anchor for alert (SE91).
            TOOL_COLLECTING             = 'USER_CHECK'
                                        " Logical name of program
                                        " to run to collect data,
                                        " defined in RZ21.
*           tool_onalert                = 'TELL_ADMIN_TOO_MANY_USERS'
                                        " Logical name of program
                                        " to start if an alert is
                                        " triggered.
            TOOL_ANALYZE                = 'Active_Users'
                                        " Tool for analyzing alerts
                                        " or current condition.
            TOOL_TOOLDISPATCHER         = 'SAP_CCMS_DEFAULT_TD'
                                        " Alert monitor checks
                                        " every 5 minutes to see if
                                        " collection tool should
                                        " run. If so, starts tool
                                        " in dialog work process.
                                        " Alternative:
                                        " SAP_CCMS_BATCH_DISPATCHER
                                        " Run collection tool in
                                        " batch instead of dialog.
                                        " Leave out to run tool
                                        " under your own control.
       IMPORTING
            NEW_TID                     = MA_P_TID "MAttribute TID
       EXCEPTIONS
            INVALID_TID                 = 1
            UNABLE_TO_EXPAND_NAME       = 2
            INVALID_PARAMETERS          = 3
            COMMUNICATION_FAILURE       = 4
            OTHER_PROBLEM               = 5
            WRONG_SEGMENT               = 6
            INTERNAL_FAILURE_SALS       = 7
            OTHERS                      = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

ENDFORM.


*&---------------------------------------------------------------------*
*&      Form  report_sample_value
*&---------------------------------------------------------------------*
FORM REPORT_SAMPLE_VALUE USING
                           MA_P_TID LIKE ALGLOBTID
                           EX_P_VALUE TYPE I.
```

```
  call function 'SALI_PERF_REPORT_VALUE'
     exporting
          TOTAL_OF_REPORTED_VALUES  = EX_P_VALUE
          NUMBER_OF_REPORTED_VALUES = 1
          REPORTEDBY                = 'RSDSSMPL_PERFORMANCE'
     changing
          TID                       = MA_P_TID
     exceptions
          invalid_tid               = 1
          wrong_typeclass           = 2
          invalid_parameters        = 3
          communication_failure     = 4
          other_problem             = 5
          others                    = 6.

  if sy-subrc <> 0.
  endif.

ENDFORM.
```

Sample Code: Status Message Collection Method

```
*&---------------------------------------------------------------------*
*& Report RSDSSMPL_STATUS:  Template for a data collection method      *
*&                 for passing status or error (T100) messages         *
*&                 to the Release 4 Alert Monitor.                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*
REPORT   RSDSSMPL_STATUS.

*---------------------------------------------------------------------*

* How the report works:  A quick summary.
* The report sees to it that your component is represented in the
* alert monitor.  It creates a monitoring object and monitoring
* attribute in the alert monitor.  The monitoring object represents
* your component.  The monitoring attribute represents the status
* message(s) that you are going to report for your component.  Each
* time the report runs, it creates the object and attribute if they
* are not yet available, or attaches to them, if they already have
* been created.  The report then passes the latest status message
* from your component into the monitoring attribute.  The alert
* monitor then displays your message, triggers an alert if necessary,
* and so on.

* Embedding the report in your application: You will need to do the
* activities represented by this report every time that your component
* issues a significant message (one that should be reported in the
* alert monitor).  It would therefore be best to implement the report
* as a function module that you can call every time that your
* component issues a significant message.  Just pass the message
* ID and number (from table T100) into the function module, together
* with values for any variables in the message text and the alert
* criticality of the message, and your status message monitoring
* is functional.

* Note:  You do not need to report system log messages as status
* messages.  System log messages are already reported to the alert
* monitor in system log message containers.

*---------------------------------------------------------------------*

* Constants for the Release 4 Monitoring Architecture.
INCLUDE RSALEXTI.

* Constants for use in this sample program.
CONSTANTS:
* Monitoring object constants...
           SAMPLE_OBJECT_NAME VALUE 'DemoObject: Batch Input'
              LIKE ALMTCUSGEN-MTNAMESHRT,
              " Name of the monitoring object to create and access in
              " the alert monitor. Enter the name that you wish to
              " display to users in the alert monitor.
              "
              " This sample report creates a new monitoring object
              " under the context "<System ID>/<Instance name>". The
              " object appears in the alert monitoring tree at the same
              " level as such static MTEs (Monitoring Tree Elements,
              " a generic term for alert monitor nodes) such as
              " "OperatingSystem" or "R3Services". The first run of the
```

```
              " program creates the object. Subsequent runs "attach" to
              " the newly-created object.

            SAMPLE_OBJECT_CLASS  VALUE 'SampleObjectClass'
               LIKE ALGRPCUSGE-CUSGRPNAME,
               " MTE class for the monitoring object: Enter a freely-
               " definable class name for certain customizing attributes
               " of your object.
               "
               " The MTE class lets you group the General Properties
               " Tool Specifications (TX RZ21) settings for an object.
               " Use: Lets all instances of an object share by reference
               " the same general properties and tool assignments.

* Monitoring attribute constants...
            SAMPLE_ATTRIBUTE_NAME VALUE
               'Sample Value(Batch Input Session Prep)'
               LIKE ALMTCUSGEN-MTNAMESHRT,
               " Name of the monitoring attribute to be created for the
               " object above in sample_object_name. Enter the name that
               " you wish to display in the alert monitor. You report
               " information on an object only through the object's
               " monitoring attributes.
                                                          "
               " This sample collection tool creates a status
               " attribute for reporting the occurrence of a message.
               " You can report any message pertaining to status to
               " the attribute (any of various types of error, for
               " example) or allocate an attribute to a particular
               " message.

            SAMPLE_ATTRIBUTE_CLASS   VALUE  'SampleStatusAttribClass'
               LIKE ALGRPCUSGE-CUSGRPNAME,
               " MTE class for the monitoring attribute: Enter a freely-
               " definable class name for the general properties and
               " method assignments of your MA.

            SAMPLE_CUSTOMIZING_GROUP VALUE 'SampleStatusGroup'
               LIKE ALGRPCUSGE-CUSGRPNAME.
              " Customizing group for the monitoring attribute.  Enter
              " a freely-definable name for the alert thresholds of
              " your monitoring attribute.
              "
              " A customizing group is an attribute-only complement
              " to the MTE class. The customizing group lets you store
              " attribute-specific customizing settings: the alert-
              " handling for the monitoring attribute (TX RZ21).
              " Monitoring attributes of the same type, such as
              " instances of logon errors in different systems, can
              " share by reference common alert-handling settings.

*--  global variables of program --*
DATA: SAMPLE_STATUS_TID LIKE ALGLOBTID,
      " This is the TID (handle) in the alert monitor of the
      " monitoring object that this report will create and use.
      SAMPLE_STATUS_CRITICALITY LIKE ALALERTRC-VALUE
                                VALUE AL_VAL_YELLOW,
      " The criticality (green, yellow, red) of a message that is
      " being reported into the alert monitor
      SAMPLE_T100MESSAGE_ID LIKE SY-MSGID VALUE 'RT',
      " The message ID of a message to report
      SAMPLE_T100MESSAGE_NUMBER LIKE SY-MSGNO VALUE '021',
      " The number of a message to report
      SAMPLE_MSGARGTYPE_1 LIKE SXMIMSGRAW-ARGTYPE1 VALUE 'C',
      SAMPLE_VARIABLE_1(12) TYPE C VALUE 'Batch input ',
```

```
      " Argument type and value for possible variables in T100 messages
      SAMPLE_MSGARGTYPE_2 LIKE SXMIMSGRAW-ARGTYPE1 VALUE 'C',
      SAMPLE_VARIABLE_2(20) TYPE C VALUE 'transfer program ',
      SAMPLE_MSGARGTYPE_3 LIKE SXMIMSGRAW-ARGTYPE1 VALUE 'C',
      SAMPLE_VARIABLE_3(15) TYPE C VALUE 'sample_program ',
      SAMPLE_MSGARGTYPE_4 LIKE SXMIMSGRAW-ARGTYPE1 VALUE 'C',
      SAMPLE_VARIABLE_4(22) TYPE C VALUE 'completed with errors.'.

*--  Start of the main program --*
START-OF-SELECTION.

* Step 1: Create/attach monitoring object and monitoring attribute.
* The first program run creates the object and attribute (makes them
* visible in the alert monitor). Subsequent program runs simply
* attach to the object, since it has already been created. Each
* program run also reports a new message.  The new message replaces
* any previous messages as the "current status" shown in the
* attribute.  Of course, any alerts caused by previous messages
* are unaffected and remain available.
  PERFORM CREATE_MO_MA_EXAMPLE CHANGING SAMPLE_STATUS_TID.

* Step 2: Report the status message into the alert monitor.  If you are
* not sure where the message came from, see "embedding this report"
* in your application, at the start of the code.
  PERFORM REPORT_SAMPLE_MSG USING SAMPLE_STATUS_TID
                                  SAMPLE_STATUS_CRITICALITY
                                  SAMPLE_T100MESSAGE_ID
                                  SAMPLE_T100MESSAGE_NUMBER
                                  SAMPLE_MSGARGTYPE_1
                                  SAMPLE_VARIABLE_1
                                  SAMPLE_MSGARGTYPE_2
                                  SAMPLE_VARIABLE_2
                                  SAMPLE_MSGARGTYPE_3
                                  SAMPLE_VARIABLE_3
                                  SAMPLE_MSGARGTYPE_4
                                  SAMPLE_VARIABLE_4.

********************** end of the main program

*---------------------------------------------*
*-- create_mo_ma_example ---------------------*
*---------------------------------------------*
*
* Create/attach to the monitoring object and monitoring attribute
*
FORM CREATE_MO_MA_EXAMPLE USING MA_P_TID LIKE ALGLOBTID.

  DATA: PARENT_TID LIKE ALGLOBTID,
        " This is the TID (handle) of the element in the monitoring
        " tree under which our new monitoring object and attribute
        " are to appear. Here, the parent is the current server.
        MO_TID     LIKE ALGLOBTID.
        " This is the TID of the new monitoring object created and
        " accessed by this program.

* Step 1: Get the TID of the parent element in the alert monitor tree
  call function 'SALI_MT_GET_TID_BY_NAME'
    EXPORTING
      MT_FULL_NAME          = '\&SY\&INSTANCE_NAME'
                              " Supported variables:
                              " &SY: R/3 System name - C11
                              " &N: SAP System number - 53
                              " &HOSTNAM: Host name as in system
                              "    profile parameter saplocalhost-
                              "    host1
```

```
                              " &INSTANCE_NAME: Instance name -
                              "    profile parameter rdisp/myname-
                              "    host1_C11_53
    IMPORTING
      TID                   = PARENT_TID
    EXCEPTIONS
      UNABLE_TO_EXPAND_NAME = 1
      NAME_NOT_FOUND        = 2
      COMMUNICATION_FAILURE = 3
      OTHER_PROBLEM         = 4
      OTHERS                = 5.

* Step 2: Create a new monitoring object / attach to the existing
* monitoring object under parent_tid.
  CALL FUNCTION 'SALI_MO_CREATE_ATTACH'
    EXPORTING
      PARENT_TID            = PARENT_TID
      MO_NAME               = SAMPLE_OBJECT_NAME
      NUMRANGE              = AL_NR_TEMP " temporary number range
                              " Sample uses the temporary number
                              " range for the MO handle. Benefit:
                              " the monitoring architecture
                              " manages the number range; you
                              " don't have to. See documentation
                              " on number ranges.
      UNIQUENUM             = 0
                              " ID number. Set to 0 if using
                              " numrange = al_nr_temp.
      VISIBLE_ON_USERLEV    = AL_VISIBLE_OPERATOR
                              " Specify level for display in
                              " alert monitor:
                              " al_visible_operator: Always
                              "   visible.
                              " al_visible_expert: Only visible
                              "   in higher Analysis view
                              " al_visible_developer: Only
                              "   visible in Expert Analysis view
                              " Note: Alerts are always visible.
      DESC_TEXT_MSGID       = 'RT'
      DESC_TEXT_MSGNO       = 094
                              " T100-Message: Display as descript.
                              " and F1-help anchor for MO (SE91)
      MT_CLASS              = SAMPLE_OBJECT_CLASS
    IMPORTING
      NEW_TID               = MO_TID
    EXCEPTIONS
      INVALID_TID           = 1
      UNABLE_TO_EXPAND_NAME = 2
      INVALID_PARAMETERS    = 3
      COMMUNICATION_FAILURE = 4
      OTHER_PROBLEM         = 5
      WRONG_SEGMENT         = 6
      INTERNAL_FAILURE_SALS = 7
      OTHERS                = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.


* Step 3: Create the status message monitoring attribute (MA)
  CALL FUNCTION 'SALI_SMES_CREATE_ATTACH'
    EXPORTING
      PARENT_TID             = MO_TID
      MTE_NAME               = SAMPLE_ATTRIBUTE_NAME
```

```
MTE_CLASS              = SAMPLE_ATTRIBUTE_CLASS
MTE_NUMRANGE           = AL_NR_TEMP " See above.
MTE_UNIQUENUM          = 0          " See above.
MTE_SEVERITY           = 50
                         " Value from 0 (minor) to 255 (severe)
                         " that shows how important an alert in
                         " the MA is. Ex: Too many users is not
                         " very important.  Failed updates are.
                         " See documentation on screening messages
                         " and ranking alerts.
MTE_VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR
                         " See above.
MTE_KEEPALTYPE         = AL_KEEP_ALL
                         " Keep alerts? Default: keep all to
                         " limit in mte_keepalmax.  Others:
                         " al_keep_oldest
                         " al_keep_newest
                         " al_keep_highest
                         " al_keep_slave
MTE_KEEPALMAX          = 10
                         " Keep a maximum of 10 alerts. Discard
                         " excess according to keepaltype.
MTE_SECONDS_TIL_COLLECTINGTOOL = 0
                         " How often should alert monitor
                         " start collection tool?  Here,
                         " our program reports messages
                         " as needed.  The value 0 tells
                         " the alert monitor not to start
                         " the collection tool.
MTE_SECONDS_UNTIL_SET_INACTIVE = 0
                         " Is element active? Set to 0 (off)
                         " because our program reports status
                         " at unpredictable intervals.
MTE_SECONDS_WARMUPTIME   = 0
                         " Ignore alerts for this time period
                         " (sec.) during warm-up.
MTE_F1_HELP_TEXT_MSGID = 'RT'
MTE_F1_HELP_TEXT_MSGNO = 093
                         " T100-Message to display as descript.
                         " and F1-help anchor for MA (SE91).
SMES_CUSTOMIZING_GROUP = SAMPLE_CUSTOMIZING_GROUP
SMES_SUBTYPE           = AL_STD_NO_SUBCLASS " Default value
                         " Monitor behavior: No alert implies
                         " value "Green".  New or cleared
                         " attribute is green in the monitor.
                         " Alternate:
                         " al_std_smes_green_only_explicit
                         "   No alert: Attribute is inactive,
                         "   color white in monitor.  Attribute
                         "   is set to green only by reporting
                         "   of message with green criticality.
SMES_ALERT_MODE        = AL_SMSG_ALMODE_ALWAYS
                         " Modify alert-triggering behavior.
                         " Here:
                         " Always trigger an alert if a message
                         " with yellow or red criticality is
                         " reported (See also RZ21).
                         " Alternatives:
                         " al_smsg_almode_value_chg
                         "   Trigger alert only on increase in
                         "   message criticality (e.g. yellow
                         "   to red) or if no alert exists yet.
                         " al_smsg_almode_msg_chg:  Trigger an alert
                         "   only if the reported message, the criticality,
                         "   or the severity changes and the message
```

```
                                " criticality warrants an alert.
                                " al_smsg_almode_never
                                "   Never trigger an alert, only
                                "   report (and display) messages.
      SMES_ALERT_SHIFT       = AL_SMSG_ALSHIFT_UNCHG
                                " Shift criticality of alerts. Here:
                                " Use the criticality of the reported
                                " message to determine the criticality
                                " of the alert that is triggered.
                                " Alternatives:
                                " al_smsg_alshift_r_as_y
                                "   No red alerts. Red becomes yellow.
                                " al_smsg_alshift_y_as_r
                                "   No yellow alerts. Yellow becomes
                                "   red.
                                " al_smsg_alshift_ray_yag
                                "   Downgrade all alerts. Red becomes
                                "   yellow; yellow becomes green.
*     tool_collecting        = 'BATCH_INPUT_STATUS'
                                " Logical name of program
                                " to run to collect data,
                                " defined in RZ21. Here, can be omitted
                                " as this data supplier is started
                                " by the monitored component.
*     tool_onalert           = 'TELL_ADMIN_BATCH_INPUT_ERROR'
                                 " Logical name of program
                                 " to start if an alert is
                                 " triggered.
*     tool_analyze           = 'BATCH_INPUT_MANAGEMENT'
                                " Tool for analyzing alerts
                                " or current condition.
      TOOL_TOOLDISPATCHER    = 'SAP_CCMS_DEFAULT_TD'
                                " Not relevant, can be omitted, since
                                " this data supplier is started by
                                " the monitored component.
    IMPORTING
             NEW_TID                       = MA_P_TID "MAttribute TID
        EXCEPTIONS
             INVALID_TID                   = 1
             UNABLE_TO_EXPAND_NAME         = 2
             INVALID_PARAMETERS            = 3
             COMMUNICATION_FAILURE         = 4
             OTHER_PROBLEM                 = 5
             WRONG_SEGMENT                 = 6
             INTERNAL_FAILURE_SALS         = 7
             OTHERS                        = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

ENDFORM.


*&---------------------------------------------------------------------*
*&      Form  report_sample_msg
*&---------------------------------------------------------------------*
*
* Report a status message into the monitoring architecture and alert
* monitor.
*
FORM REPORT_SAMPLE_MSG USING
                        MA_M_TID LIKE ALGLOBTID
                        MSG_VAL LIKE SAMPLE_STATUS_CRITICALITY
                        MSG_ID LIKE SAMPLE_T100MESSAGE_ID
```

```
                        MSG_NO LIKE SAMPLE_T100MESSAGE_NUMBER
                        TYPE_1 LIKE SAMPLE_MSGARGTYPE_1
                        VAL_1 LIKE SAMPLE_VARIABLE_1
                        TYPE_2 LIKE SAMPLE_MSGARGTYPE_2
                        VAL_2 LIKE SAMPLE_VARIABLE_2
                        TYPE_3 LIKE SAMPLE_MSGARGTYPE_3
                        VAL_3 LIKE SAMPLE_VARIABLE_3
                        TYPE_4 LIKE SAMPLE_MSGARGTYPE_4
                        VAL_4 LIKE SAMPLE_VARIABLE_4.

  CALL FUNCTION 'SALI_SMES_REPORT_T100_MESSAGE'
       EXPORTING
            MSGVALUE              = MSG_VAL
                                    " Criticality of the message:
                                    " al_val_green: Success message.
                                    " al_val_yellow: Warning message.
                                    " al_val_red: Error/Problem message.
            MSGID                 = MSG_ID
            MSGNO                 = MSG_NO
                                    " Message ID and number from table
                                    " T100.
            MSGARG1               = VAL_1
*            argtype1              = type_1
                                    " Parameter type and value for the
                                    " optional parameters in T100
                                    " messages.
            MSGARG2               = VAL_2
*            argtype2              = type_2
            MSGARG3               = VAL_3
*            argtype3              = type_3
            MSGARG4               = VAL_4
*            argtype4              = type_4
            DEFAULT_MSGTEXT       = ''
                                    " Default text to display if no
                                    " T100 message in English or German
                                    " is found.  Important:  Leave this
                                    " field empty or omit it if your
                                    " T100 message has been correctly
                                    " created in the system.  If you
                                    " specify a default text, then it
                                    " will be used instead of the text
                                    " of the T100 message that you
                                    " report.
            REPORTEDBY            = 'RSDSSMPL_STATUS'
       CHANGING
            TID                   = MA_M_TID
       EXCEPTIONS
            INVALID_TID           = 1
            WRONG_TYPECLASS       = 2
            INVALID_PARAMETERS    = 3
            COMMUNICATION_FAILURE = 4
            OTHER_PROBLEM         = 5
            WRONG_SEGMENT         = 6
            INTERNAL_FAILURE_SALS = 7
            OTHERS                = 8.

  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

  if sy-subrc <> 0.
  endif.

ENDFORM.
```

Sample Code: Message Log Collection Method

```
*&---------------------------------------------------------------------*
*& Report RSDSSMPL_LOG:  Template for a data collection method         *
*&                 for passing log or trace (T100) messages            *
*&                 to the Release 4 Alert Monitor.                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*
REPORT   RSDSSMPL_LOG.

*---------------------------------------------------------------------*

* How the report works:  A quick summary.
* The report sees to it that your component is represented in the
* alert monitor.  It creates a monitoring object and monitoring
* attribute in the alert monitor.  The monitoring object represents
* your component.  The monitoring attribute represents the log/trace
* message(s) that you are going to report for your component.  Each
* time the report runs, it creates the object and attribute if they
* are not yet available, or attaches to them, if they already have
* been created.  The report then passes the latest message
* from your component into the monitoring attribute.  The alert
* monitor then displays your message, triggers an alert if necessary,
* and so on.

* Embedding the report in your application: You will need to do the
* activities represented by this report every time that your component
* adds one or more messages to the log.  It would therefore be best to
* implement the report as a function module that you can call every time
* that your component issues a significant message.  Just pass the msg
* ID and number (from table T100) into the function module, together
* with values for any variables in the message text and the alert
* criticality of the message, and your message log monitoring
* is functional.

* Note:  You do not need to report system log messages as log
* messages.  System log messages are already reported to the alert
* monitor in system log message containers.

*---------------------------------------------------------------------*

* Constants for the Release 4 Monitoring Architecture.
INCLUDE RSALEXTI.

* Constants for use in this sample program.
CONSTANTS:
* Monitoring object constants...
           SAMPLE_OBJECT_NAME VALUE 'DemoObject: Message Log'
              LIKE ALMTCUSGEN-MTNAMESHRT,
              " Name of the monitoring object to create and access in
              " the alert monitor. Enter the name that you wish to
              " display to users in the alert monitor.
              "
              " This sample report creates a new monitoring object
              " under the context "<System ID>/<Instance name>". The
              " object appears in the alert monitoring tree at the same
              " level as such static MTEs (Monitoring Tree Elements,
              " a generic term for alert monitor nodes) such as
              " "OperatingSystem" or "R3Services". The first run of the
              " program creates the object. Subsequent runs "attach" to
```

```
              " the newly-created object.

            SAMPLE_OBJECT_CLASS  VALUE 'SampleMTObjectClass'
               LIKE ALGRPCUSGE-CUSGRPNAME,
               " MTE class for the monitoring object: Enter a freely-
               " definable class name for certain customizing attributes
               " of your object.
               "
               " The MTE class lets you group the General Properties
               " Tool Specifications (TX RZ21) settings for an object.
               " Use: Lets all instances of an object share by reference
               " the same general properties and tool assignments.

* Monitoring attribute constants...
            SAMPLE_ATTRIBUTE_NAME VALUE
               'Sample Value(My Component''s log)'
               LIKE ALMTCUSGEN-MTNAMESHRT,
               " Name of the monitoring attribute to be created for the
               " object above in sample_object_name. Enter the name that
               " you wish to display in the alert monitor. You report
               " information on an object only through the object's
               " monitoring attributes.
               "
               " This sample collection tool creates a log
               " attribute for keeping a log or trace of messages.
               " You can structure logs any way you wish: report all
               " messages from your component to the the attribute, or
               " screen them, or assign different types of messages to
               " different message containers for your component (object)

            SAMPLE_ATTRIBUTE_CLASS   VALUE  'SampleLogAttribClass'
               LIKE ALGRPCUSGE-CUSGRPNAME,
               " MTE class for the monitoring attribute: Enter a freely-
               " definable class name for the general properties and
               " method assignments of your MA.

            SAMPLE_CUSTOMIZING_GROUP VALUE 'SampleLogGroup'
               LIKE ALGRPCUSGE-CUSGRPNAME.
              " Customizing group for the monitoring attribute.  Enter
              " a freely-definable name for the alert thresholds of
              " your monitoring attribute.
              "
              " A customizing group is an attribute-only complement
              " to the MTE class. The customizing group lets you store
              " attribute-specific customizing settings: the alert-
              " handling for the monitoring attribute (TX RZ21).
              " Monitoring attributes of the same type, such as
              " instances of logon errors in different systems, can
              " share by reference common alert-handling settings.

*--  global variables of program --*
DATA: SAMPLE_LOG_TID LIKE ALGLOBTID,
      " This is the TID (handle) in the alert monitor of the
      " monitoring object that this report will create and use.
      SAMPLE_LOG_CRITICALITY LIKE ALALERTRC-VALUE,
      " The criticality (green, yellow, red) of a message that is
      " being reported into the alert monitor
      SAMPLE_T100MESSAGE_ID LIKE SY-MSGID VALUE 'RA',
      " The message ID of a message to report
      SAMPLE_T100MESSAGE_NUMBER LIKE SY-MSGNO,
      " The number of a message to report
      SAMPLE_MSGARGTYPE_1 LIKE SXMIMSGRAW-ARGTYPE1,
      SAMPLE_VARIABLE_1(12) TYPE C,
      " Argument type and value for possible variables in T100 messages
      SAMPLE_MSGARGTYPE_2 LIKE SXMIMSGRAW-ARGTYPE1,
```

```
      SAMPLE_VARIABLE_2(20) TYPE C,
      SAMPLE_MSGARGTYPE_3 LIKE SXMIMSGRAW-ARGTYPE1,
      SAMPLE_VARIABLE_3(15) TYPE C,
      SAMPLE_MSGARGTYPE_4 LIKE SXMIMSGRAW-ARGTYPE1,
      SAMPLE_VARIABLE_4(22) TYPE C.

*--  Start of the main program --*
START-OF-SELECTION.

* Step 1: Create/attach monitoring object and monitoring attribute.
* The first program run creates the object and attribute (makes them
* visible in the alert monitor). Subsequent program runs simply
* attach to the object, since it has already been created. Each
* program run also reports a new message.  The new message is
* independently evaluated as the potential trigger for an alert
* and is added to the message log.
  PERFORM CREATE_MO_MA_EXAMPLE CHANGING SAMPLE_LOG_TID.

* Step 2: Report one or more messages into the alert monitor.  If you
* are not sure where the message came from, see "embedding this report"
* in your application, at the start of the code.  Here, we simulate
* a log by looping through a list of messages -- to demonstrate log
* attribute functionality.

* loop variables
DATA: BEGIN OF MSGTAB OCCURS 3,
  MESSAGE_NO(3) TYPE C,
  MESSAGE_CRIT(14) TYPE C,
  MESSAGE_VAR1(20) TYPE C,
END OF MSGTAB.


  MSGTAB-MESSAGE_NO = '144'. MSGTAB-MESSAGE_CRIT = AL_VAL_YELLOW.
  MSGTAB-MESSAGE_VAR1 = 'Special_Monitor'.
  APPEND MSGTAB.

  MSGTAB-MESSAGE_NO = '140'. MSGTAB-MESSAGE_CRIT = AL_VAL_RED.
  MSGTAB-MESSAGE_VAR1 = ''.
  APPEND MSGTAB.

  MSGTAB-MESSAGE_NO = '143'. MSGTAB-MESSAGE_CRIT = AL_VAL_GREEN.
  MSGTAB-MESSAGE_VAR1 = ''.
  APPEND MSGTAB.

LOOP AT MSGTAB.

  SAMPLE_LOG_CRITICALITY = MSGTAB-MESSAGE_CRIT.
  SAMPLE_T100MESSAGE_NUMBER = MSGTAB-MESSAGE_NO.
  SAMPLE_VARIABLE_1 = MSGTAB-MESSAGE_VAR1.

  PERFORM REPORT_SAMPLE_MSG USING SAMPLE_LOG_TID
                                  SAMPLE_LOG_CRITICALITY
                                  SAMPLE_T100MESSAGE_ID
                                  SAMPLE_T100MESSAGE_NUMBER
                                  SAMPLE_MSGARGTYPE_1
                                  SAMPLE_VARIABLE_1
                                  SAMPLE_MSGARGTYPE_2
                                  SAMPLE_VARIABLE_2
                                  SAMPLE_MSGARGTYPE_3
                                  SAMPLE_VARIABLE_3
                                  SAMPLE_MSGARGTYPE_4
                                  SAMPLE_VARIABLE_4.

ENDLOOP.
********************** end of the main program
```

```
*---------------------------------------------*
*-- create_mo_ma_example ---------------------*
*---------------------------------------------*
*
* Create/attach to the monitoring object and monitoring attribute
*
FORM CREATE_MO_MA_EXAMPLE USING MA_P_TID LIKE ALGLOBTID.

  DATA: PARENT_TID LIKE ALGLOBTID,
        " This is the TID (handle) of the element in the monitoring
        " tree under which our new monitoring object and attribute
        " are to appear. Here, the parent is the current server.
        MO_TID     LIKE ALGLOBTID.
        " This is the TID of the new monitoring object created and
        " accessed by this program.

* Step 1: Get the TID of the parent element in the alert monitor tree
  call function 'SALI_MT_GET_TID_BY_NAME'
    EXPORTING
      MT_FULL_NAME          = '\&SY\&INSTANCE_NAME'
                              " Supported variables:
                              " &SY: R/3 System name - C11
                              " &N: SAP System number - 53
                              " &HOSTNAM: Host name as in system
                              "    profile parameter saplocalhost-
                              "    host1
                              " &INSTANCE_NAME: Instance name -
                              "    profile parameter rdisp/myname-
                              "    host1_C11_53
    IMPORTING
      TID                   = PARENT_TID
    EXCEPTIONS
      UNABLE_TO_EXPAND_NAME = 1
      NAME_NOT_FOUND        = 2
      COMMUNICATION_FAILURE = 3
      OTHER_PROBLEM         = 4
      OTHERS                = 5.

* Step 2: Create a new monitoring object / attach to the existing
* monitoring object under parent_tid.
  CALL FUNCTION 'SALI_MO_CREATE_ATTACH'
    EXPORTING
      PARENT_TID            = PARENT_TID
      MO_NAME               = SAMPLE_OBJECT_NAME
      NUMRANGE              = AL_NR_TEMP " temporary number range
                              " Sample uses the temporary number
                              " range for the MO handle. Benefit:
                              " the monitoring architecture
                              " manages the number range; you
                              " don't have to. See documentation
                              " on number ranges.
      UNIQUENUM             = 0
                              " ID number. Set to 0 if using
                              " numrange = al_nr_temp.
      VISIBLE_ON_USERLEV    = AL_VISIBLE_OPERATOR
                              " Specify level for display in
                              " alert monitor:
                              " al_visible_operator: Always
                              "   visible.
                              " al_visible_expert: Only visible
                              "   in higher Analysis view
                              " al_visible_developer: Only
                              "   visible in Expert Analysis view
                              " Note: Alerts are always visible.
```

```
      DESC_TEXT_MSGID       = 'RT'
      DESC_TEXT_MSGNO       = 092
                              " T100-Message: Display as descript.
                              " and F1-help anchor for MO (SE91)
      MT_CLASS              = SAMPLE_OBJECT_CLASS
    IMPORTING
      NEW_TID               = MO_TID
    EXCEPTIONS
      INVALID_TID           = 1
      UNABLE_TO_EXPAND_NAME = 2
      INVALID_PARAMETERS    = 3
      COMMUNICATION_FAILURE = 4
      OTHER_PROBLEM         = 5
      WRONG_SEGMENT         = 6
      INTERNAL_FAILURE_SALS = 7
      OTHERS                = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.


* Step 3: Create the status message monitoring attribute (MA)
  CALL FUNCTION 'SALI_MSC_CREATE_ATTACH'
    EXPORTING
      PARENT_TID             = MO_TID
      MTE_NAME               = SAMPLE_ATTRIBUTE_NAME
      MTE_CLASS              = SAMPLE_ATTRIBUTE_CLASS
      MTE_NUMRANGE           = AL_NR_TEMP " See above.
      MTE_UNIQUENUM          = 0          " See above.
      MTE_SEVERITY           = 50
                               " Value from 0 (minor) to 255 (severe)
                               " that shows how important an alert in
                               " the MA is. Ex: Too many users is not
                               " very important.  Failed updates are.
                               " See documentation on screening messages
                               " and ranking alerts.
      MTE_VISIBLE_ON_USERLEV = AL_VISIBLE_OPERATOR
                               " See above.
      MTE_KEEPALTYPE         = AL_KEEP_ALL
                               " Keep alerts? Default: keep all to
                               " limit in mte_keepalmax.  Others:
                               " al_keep_oldest
                               " al_keep_newest
                               " al_keep_highest
                               " al_keep_slave
      MTE_KEEPALMAX          = 10
                               " Keep a maximum of 10 alerts. Discard
                               " excess according to keepaltype.
      MTE_SECONDS_TIL_COLLECTINGTOOL = 0
                               " How often should alert monitor
                               " start collection tool?  Here,
                               " our program reports messages
                               " as needed.  The value 0 tells
                               " the alert monitor not to start
                               " the collection tool.
      MTE_SECONDS_UNTIL_SET_INACTIVE = 0
                               " Is element active? Set to 0 (off)
                               " because our program reports status
                               " at unpredictable intervals.
      MTE_SECONDS_WARMUPTIME = 0
                               " Ignore alerts for this time period
                               " (sec.) during warm-up.
      MTE_F1_HELP_TEXT_MSGID = 'RT'
      MTE_F1_HELP_TEXT_MSGNO = 141
```

```
                              " T100-Message to display as descript.
                              " and F1-help anchor for MA (SE91).
      MSC_CUSTOMIZING_GROUP  = SAMPLE_CUSTOMIZING_GROUP
      MSC_SUBTYPE            = AL_STD_MSC_CACHE " Default value
                              " Behavior: Monitoring architecture
                              " provides log functionality. Messages
                              " are held in an internal cache
                              " provided by the monitoring
                              " architecture.  See log type docu.
                              " Alternates:
                              " al_std_msc_syslog
                              "   The R/3 system log provides the
                              "   message log functionality. Only
                              "   alerts are held by the monitoring
                              "   architecture, message are read from
                              "   the system log for display.
                              " al_std_msc_external
                              "   A log facility external to the
                              "   monitoring architecture provides
                              "   the log functionality (not the
                              "   R/3 system log). You must provide
                              "   the function call for retrieving
                              "   messages from the log for display.
                              " At this time, only "cache" is allowed.
      MSC_RAISE_VALUE        = AL_VAL_GREEN " Default
                              " With msc_raise_severity, determines
                              " when alerts can be triggered by
                              " messages in message logs.  "green"
                              " lets all messages with criticality
                              " yellow or red trigger alerts.
                              " 'al_value_yellow' would allow only
                              " messages with criticality yellow or red
                              " & severity higher than msc_raise_sever.
                              " to trigger alerts. Values defined
                              " in rsalexti.
      MSC_RAISE_SEVERITY     = 255 " Default
                              " With msc_raise_value, determines
                              " when alerts can be triggered by
                              " messages in message logs.  Here,
                              " only messages with more than
                              " green/255 trigger alerts (any
                              " yellow or red, no green messages).
                              " See also message screening docu.
      MSC_ACTUAL_MSG_MODE    = AL_MSC_VAL_MODE_LAST " Default
                              " Determines which message should be
                              " displayed as the current status msg
                              " in the alert monitor.  Here, the
                              " most recent message is displayed.
                              " Alternatives:
                              " al_td_msc_val_mode_highalrt:  Display
                              "   the most important alert.
                              " al_td_msc_val_mode_worst_since: Display
                              "   most important alert (if any) since
                              "   during the last X minutes. X is
                              "   specified in msc_actual_msg_time.
      MSC_ACTUAL_MSG_TIME    = 20
                              " Omit unless mode is "worst_since".
                              " Then specifies "find worst message
                              " in the last actual_msg_time minutes."
*     tool_collecting        = 'BATCH_INPUT_STATUS'
                              " Logical name of program
                              " to run to collect data,
                              " defined in RZ21.
*     tool_onalert           = 'TELL_ADMIN_BATCH_INPUT_ERROR'
                               " Logical name of program
```

```
                                " to start if an alert is
                                " triggered.
*     tool_analyze           = 'BATCH_INPUT_MANAGEMENT'
                               " Tool for analyzing alerts
                               " or current condition.
      TOOL_TOOLDISPATCHER    = 'SAP_CCMS_DEFAULT_TD'
                               " Not relevant, can be omitted, since
                               " this data supplier is started by
                               " the monitored component.
    IMPORTING
             NEW_TID         = MA_P_TID "MAttribute TID
        EXCEPTIONS
             INVALID_TID                    = 1
             UNABLE_TO_EXPAND_NAME          = 2
             INVALID_PARAMETERS             = 3
             COMMUNICATION_FAILURE          = 4
             OTHER_PROBLEM                  = 5
             WRONG_SEGMENT                  = 6
             INTERNAL_FAILURE_SALS          = 7
             OTHERS                         = 8.
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

ENDFORM.


*&---------------------------------------------------------------------*
*&      Form  report_sample_msg
*&---------------------------------------------------------------------*
*
* Report a status message into the monitoring architecture and alert
* monitor.
*
* In this example, arbitrary message criticalities and severities are
* used.  In a production program, you could assign such attributes to
* messages here, for example, with a case select over message
* number, user, client, or any other criteria available from the
* application that has issued the message.

FORM REPORT_SAMPLE_MSG USING
                        MA_M_TID LIKE ALGLOBTID
                        MSG_VAL LIKE SAMPLE_LOG_CRITICALITY
                        MSG_ID LIKE SAMPLE_T100MESSAGE_ID
                        MSG_NO LIKE SAMPLE_T100MESSAGE_NUMBER
                        TYPE_1 LIKE SAMPLE_MSGARGTYPE_1
                        VAL_1 LIKE SAMPLE_VARIABLE_1
                        TYPE_2 LIKE SAMPLE_MSGARGTYPE_2
                        VAL_2 LIKE SAMPLE_VARIABLE_2
                        TYPE_3 LIKE SAMPLE_MSGARGTYPE_3
                        VAL_3 LIKE SAMPLE_VARIABLE_3
                        TYPE_4 LIKE SAMPLE_MSGARGTYPE_4
                        VAL_4 LIKE SAMPLE_VARIABLE_4.

  CALL FUNCTION 'SALI_MSC_REPORT_T100_MESSAGE'
    EXPORTING
      VALUE                = MSG_VAL
                             " Criticality of the message:
                             " al_val_green: Success message.
                             " al_val_yellow: Warning message.
                             " al_val_red: Error/Problem message.
     SEVERITY              = 50 " Default is 0
                             " Importance of the message for preliminary
                             " screening in the message container, if
```

```
                              " desired. This severity makes it possible,
                              " with "value" to restrict generation of
                              " alerts.  See also MSC_RAISE_VALUE and
                              " MSC_RAISE_SEVERITY in
                              " SALI_MSC_CREATE_ATTACH.
      MANDT                 = SY-MANDT " Client in which message occurred
                              " Optional:  Can be displayed in the
                              " detail information on an attribute in
                              " the alert monitor.
      UNAME                 = SY-UNAME " User who generated message
                              " Optional:  Can be displayed in the
                              " detail information on an attribute in
                              " the alert monitor.
      EXTERNAL_MSGLINE_ID   = ' '
                              " Reserved for future use. When external
                              " logs are supported, then you would
                              " supply the ID in the external log for
                              " the message that is being added to the
                              " log attribute. The alert monitor could
                              " then use this ID to "jump to" the
                              " message in the external log for analysis.
      MSGID                 = MSG_ID
      MSGNO                 = MSG_NO
                              " Message ID and number from table
                              " T100.
      MSGARG1               = VAL_1
*     argtype1              = type_1
                              " Parameter type and value for the
                              " optional parameters in T100
                              " messages.
      MSGARG2               = VAL_2
*     argtype2              = type_2
      MSGARG3               = VAL_3
*     argtype3              = type_3
      MSGARG4               = VAL_4
*     argtype4              = type_4
      DEFAULT_MSGTEXT       = ''
                              " Default text to display if no
                              " T100 message in English or German
                              " is found.  Important:  Leave this
                              " field empty or omit it if your
                              " T100 message has been correctly
                              " created in the system.  If you
                              " specify a default text, then it
                              " will be used instead of the text
                              " of the T100 message that you
                              " report.
      REPORTEDBY            = 'RSDSSMPL_LOG'
                              " Intended for reporting the logical name
                              " of the data supplier (the name defined
                              " in TX RZ21). Here, since the data
                              " supplier is not there, we report the
                              " program name.
    CHANGING
      TID                   = MA_M_TID
    EXCEPTIONS
      INVALID_TID           = 1
      WRONG_TYPECLASS       = 2
      INVALID_PARAMETERS    = 3
      COMMUNICATION_FAILURE = 4
      OTHER_PROBLEM         = 5
      WRONG_SEGMENT         = 6
      INTERNAL_FAILURE_SALS = 7
      OTHERS                = 8.
```

```
  IF SY-SUBRC <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

  if sy-subrc <> 0.
  endif.

ENDFORM.
```