English

Fujitsu Software

DSSM V21.0/SSCM V21.0
Subsystem Management in BS2000

User Guide

Valid for:
DSSM V21.0/SSCM V21.0

November 2025

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Table of Contents

DSSM and SSCM V21.0

1 Preface

The complexity of BS2000 processing has been reduced by breaking BS2000 system down into functional units (subsystems). SSCM and DSSM are the two software products with which all subsystems are declared and managed.

The static subsystem catalog (SSMCAT) is managed with SSCM (Static Subsystem Catalog Management). The declarations required for all subsystems belonging to a configuration are stored in the subsystem catalog. SSMCAT forms the database for DSSM.

DSSM (Dynamic Subsystem Management) is the central instance in a BS2000 system for dynamic subsystem management.

1.1 Target group and summary of contents

This manual is intended for BS2000 systems support staff.

This chapter “**Preface**” provides a general overview of the manual and a list of the changes made since the previous edition of the manual.

The manual is divided into the following main chapters:

The chapter “**The subsystem concept in BS2000 OS DX**” explains concepts and describes relations that play an important role in the subsystem concept. A description of the version dependencies between BS2000, DSSM and SSCM is followed by an overview of important DSSM-compatible products included in the basic configuration of BS2000 and by brief descriptions of selected unbundled DSSM-compatible products.

The chapter “**DSSM**” describes dynamic subsystem management (DSSM):
topics dealt with include tasks and functions, management strategies and error handling, how to activate DSSM via the parameter service at startup time, and the accounting records of DSSM.
The chapter is concluded by two comprehensive examples.

The chapter “**SSCM**” describes subsystem catalog management (SSCM). Following a brief introduction, the SSCM statements are described in detail. The chapter concludes with notes on the installation of SSCM and a number of examples.

At the back of the manual you will find a list of related publications.

Other publications referred to in the text are given in the form of abbreviated titles. The full title of each of these publications can be found under “**Related publications**”.

1.2 Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available on the internet at https://bs2manuals.ts.fujitsu.com.

1.3 Notational conventions

The following notational conventions are used in this manual:

> ℹ For drawing your attention to particularly important information.

- Terms in continuous text that are to be particularly highlighted are shown in **bold**.
- In the examples, **`bold type`** indicates a user entry and this `typewriter font` is used for system outputs.
- Short names and numbers are used for references to other publications. The complete title of each publication is contained next to the number in the list of related publications at the back of the manual.

2 The subsystem concept in BS2000 OS DX

In this chapter, important terms are explained and then information is provided for

- managing subsystems with DSSM and SSCM (see "Administering subsystems with DSSM and SSCM")
- the main DSSM-compatible products in the BS2000 basic configuration (see "Overview of important DSSM-compatible products in the BS2000 basic configuration")
- selected, unbundled DSSM-compatible products (see "Overview of selected unbundled, DSSM-compatible products").

2.1 Definitions

In BS2000, both global subsystems that are valid throughout the system and local subsystems can be grouped together to form a local subsystem configuration and administered task-local in a local subsystem catalog.
Whenever reference is made in the following to subsystems, configuration and catalog, what is meant is always the global subsystems valid throughout the system, and their configuration and catalog. Local subsystems, their configuration and catalog are always identified by the word “local”.

Subsystem

Within the context of dynamic subsystem management (DSSM), a subsystem is a unit that executes a function and that can be loaded, started and terminated automatically and independently, taking into account dependency relations to other subsystems.
A subsystem may consist of a number of subsystem components.

Each subsystem is identified uniquely to DSSM by its name and version number. A subsystem is defined for DSSM by specifying its components and attributes.

The attributes of a subsystem provide information on:

- identification (name and version)
- access privileges
- loading and processing mechanisms
- environment and dependency relations
- required address spaces
- call-up preferences
- existing call entries

The relations between different subsystems are marked by

- address space separations
- link relations
- functional dependencies
- holder task sharing

Subsystem components

The term “subsystem components” is used to denote the subsystem module (program module) and the ancillary components, known as “subsystem satellites”.
The subsystem module is a component that is absolutely essential for installing the subsystem. A subsystem may be composed of a number of different modules.
Satellites for subsystems include REP files, NOREF files, message files, syntax files and information files; these files must be declared during installation and will be required to activate the subsystem.

Subsystem call entries

The subsystem call entry is the visible point of entry through which the subsystem is accessible to user tasks.
A call entry is characterized by its name, type, scope and lifetime. A subsystem may have a number of call entries with different attributes. They are administered by DSSM.

Subsystem configuration

A subsystem configuration is the set of all subsystems that are to be available in a session. Since only one subsystem configuration can be operated in the system at a given time, this configuration must contain all the subsystems that have to be available in the system at the same time.

A subsystem configuration is defined by its subsystems and their components and attributes.

A subsystem (or subsystem version) may be compatible within a given subsystem configuration and in a certain combination, i.e. the subsystem can run in the system with certain other subsystems. A subsystem may be optional or mandatory, i.e. the subsystem configuration may or may not run without this subsystem.

When the system is in operation, a given subsystem configuration may be in one of two states:

1. Load state:
   the subsystems of the configuration are currently **active**, i.e. can actually be used;
2. Declaration state:

   the subsystems or subsystem versions of the configuration are defined in the subsystem catalog, i.e. **could be activated**.

Several different load states can be derived from a declaration state.

Subsystem catalog

Static subsystem catalog management (SSCM) generates a static subsystem catalog (SSMCAT) in which the subsystem configuration is defined.
This subsystem catalog is loaded by dynamic subsystem management (DSSM) when the system is started up.
Once loaded, the static subsystem catalog becomes a dynamic subsystem catalog.
During the session the subsystem catalog can be managed by means of the DSSM commands ADD-SUBSYSTEM, MODIFY-SUBSYSTEM-PARAMETER and REMOVE-SUBSYSTEM.

Both SSCM and DSSM use the data structure of the static subsystem catalog.
The subsystem catalog can contain up to 1000 subsystems with maximum totals of
16,000 call entries, 16,000 functional dependencies and 200 address space restrictions.

2.2 Administering subsystems with DSSM and SSCM

The purpose of dynamic subsystem management (DSSM) is to reduce the complexity of BS2000 operation by dividing the operating system into functional units (subsystems).

DSSM supports the following as subsystems:

- software products (if expressly designated as DSSM-compatible, see Tables 1 as of "Overview of selected unbundled, DSSM-compatible products" and 2 as of "Overview of important DSSM-compatible products in the BS2000 basic configuration"),
- system exit routines and
- shared products.

In addition, system administration can make customized TU programs and system exit routines DSSM-compatible and activate them as subsystems.

DSSM functions distinguish between privileged subsystems (TPR subsystems), which are part of BS2000, and nonprivileged (or unprivileged) subsystems (TU subsystems), such as user programs.
DSSM also distinguishes between subsystems that are loaded in system address space and those that are loaded in user address space. TPR subsystems are always loaded in system address space, whereas TU subsystems can be loaded either in system address space or in user address space.

The SSCM subsystem enables flexible and user-friendly management of the static subsystem catalog (SSMCAT).

The SSD object

The SSD object is an ISAM file with a key length of 11 bytes. It contains the definition(s) of one or more subsystems (but not the definitions of different versions of one and the same subsystem).
Each of these definitions contains attributes, call entries, references and dependencies of the relevant subsystem, as well as information on disjunctive subsystems and the holder task.

The SSD object itself cannot be loaded. Before it can be activated, it must be linked into a static subsystem catalog using the SSCM statement ADD-CATALOG-ENTRY.

2.3 Overview of important DSSM-compatible products in the BS2000 basic configuration

MC: Memory classes (3-6) in which the product can be loaded
TU/TPR: Processor state of the subsystems (TU: nonprivileged / TPR: privileged)

Product	Function	MC	TU/TPR
ACS	Support for file access via aliases	3,4	TPR
AIDSYS	System-related part of AID (Advanced Interactive Debugger; cf. Table 2)	3,4	TPR
BINDER	Binding of a compiled source program with other object modules and LLMs to create a loadable unit	4,6	TU
CALENDAR	Creation of user-specific calendars	3,4	TPR
CCOPY			
DIV	Object-oriented access method used especially for processing unstructured data	3,4	TPR
FASTPAM	Block-oriented access method for NK4 disk files	3,4	TU/TPR
GET-TIME	Provision of the date, standardized world time and local time	4	TU
IMON	Monitor for software installation		
IMON	(IMON-BAS) Installation and registration of software	4	TPR
IMON-GPN	Support for the allocation of logical names and path names of files	4	TPR
INIT	Initialization of magnetic tapes, MTCs and floppy disks	4,6	TU/TPR
MIP	Management and output of system messages	4	TPR
NKS	Monitoring of resource reservations	4	TPR
NKV	Monitoring of mounted data volumes	4	TPR
NKISAM	Record-oriented access method for files with “DATA” block format (without separate PAM key)	4	TPR
PAMCONV	Conversion of file formats	4,6	TU
POSIX-BC	Basic configuration component of POSIX	3,4	TPR
POSPRRTS	Runtime functions for C (in TPR)	4	TPR
REWAS			
SDF	System Dialog Facility; BS2000 language interface	4	TU
SDF-CONV	Conversion of procedure formats	6	TU

SDF-P-BASYS	Basic configuration component of SDF-P (for information on SDF-P see Table 2)	4	TPR
SECOS-KRB			
SHOW-FILE	Screen display of file contents	3,4	TPR
SIR	Software installation	4	TPR
SPOOL	Input/output control for certain device families		
PRMPRES	Creation and management of print resources in BS2000 SPOOL (component of PRM)	4	TU
PRMMAN	Creation and management of print resources in BS2000 SPOOL (component of PRM)	4	TPR
SPOOL	Organization of spoolin/spoolout and management of print jobs	4	TPR
SRPMNUC	System Resources and Privilege Management (basic configuration component of SRPM; for information on SECOS see Table 2)	4	TPR
SSCM	Generation of subsystem catalogs	4,6	TU, see SSCM
SYSFILE	Support of the system files SYSLST and SYSOUT	4	TPR
TANGRAM	Allocation of related task groups to processors in accordance with performance requirements		
TANGRAM	Regulating function	3,4	TPR
TANGBAS	Management of task groups	3,4	TPR
VOLIN	Formatting and initialization of hard disks	4,5	TPR

Table 1: DSSM-compatible products in the BS2000 basic configuration

2.4 Overview of selected unbundled, DSSM-compatible products

MC: Memory classes (3-6) in which the product can be loaded
TU/TPR: Processor state of the subsystems (TU: nonprivileged / TPR: privileged)

Product	Function	MC	TP /TPR
AID	Test and diagnostics aid (Advanced Interactive Debugger)	4	TPR
AID	Symbolic and non-symbolic debugging	4	TPR
LLMAID	Information on link and load modules	4	TPR
ARCHIVE	Saving, reconstruction and transfer of data in files and job variables	4	TPR
CRTE	Runtime functions for C and COBOL	4,6	TU
DAB	Caching in BS2000 to avoid bottlenecks (software caching)	3,4	TPR
Dprint	(Distributed Print Services) Printing in computer networks		
DPRINTCL	Client part for Dprint: creation of print jobs	4	TPR
DPRINTCM	Base mechanisms: implementation of general services	4	TPR
DPRINTSV	Server part for Dprint: management of print jobs	4	TPR
DRV	Recording procedure for maintaining duplicate disks	4	TPR
EDT	Editor for SAM, ISAM and POSIX files and for elements of program libraries	4,6	TU
FDDRL	Physical data backup of disks and pubsets	4,5	TPR
HIPLEX MSCF	Creation of a computer network on the basis of the BCAM data communication network	3,4	TPR
HSMS	Backup, archival and reconstruction of data and support for data management on external storage units	4	TPR
JV	Control of jobs and programs by means of job variables	4	TPR
LMS	Library management	4,6	TU
MAREN	Management of data on archive volumes in BS2000 computer centers	3,4	TPR
PCS	Tool for systems support in ensuring the best possible setup and operation of the BS2000 system	3,4	TPR
PROP-XT	Computer center automation and user-specific problemsolving in computer centers	4	TPR
RSO	Control of the output of remote SPOOL jobs on decentralized (RSO) printers (only executable in conjunction with SPOOL)	4	TPR
SDF-A	Management and modification of the SDF user interface	6	TU

SDF-P	Procedure and variable concept, commands for procedure control, block-oriented error handling	4	TPR
SDF-P-BIF	Builtin function of SDF-P	4	TPR
SECOS	Access security control for BS2000		
GUARDS	Management of objects (guards) and evaluation of access conditions (component of GUARDS)	4	TPR
GUARDDEF	Management of standard guard as well as the attribute and objects paths	4	TPR
GUARDCOO	Management of co-ownership guard	4	TPR
SATCP	Monitoring of events and alarms (component of SAT)	4	TPR
SRPMOPT	(Component of SRPM)	4	TPR
SM2	Monitoring system for capturing and evaluating data on BS2000 performance and the level of utilization of system resources	3,4	TPR

Table 2: DSSM-compatible, unbundled products (selection)

3 DSSM

The purpose of dynamic subsystem management (DSSM) is to reduce the complexity of BS2000 execution by dividing the operating system into functional units (subsystems).

Subsystems have the following features:

- each subsystem constitutes a self-contained unit
- subsystems can be activated, suspended, resumed and deactivated during the BS2000 session
- If a new version has been created for a subsystem, the old version can be exchanged for the new one or both versions can be operated in parallel.

DSSM is the central facility for BS2000 subsystem configuration management. DSSM controls the loading, initialization, suspension, resumption and termination of subsystems during the session. DSSM can modify the current system dynamically (on-line) by incorporating subsystems in the subsystem catalog or removing them from it, without having to suspend and restart the system as a whole.

DSSM is mandatory for BS2000 since important components of the basic configuration (see Overview of important DSSM-compatible products in the BS2000 basic configuration) can be operated only with DSSM.

DSSM command privileges

As part of the BS2000 privilege concept, a system-global privilege for execution of DSSM commands exists. The privilege to carry out subsystem management is referred to in commands and messages as SUBSYSTEM-MANAGEMENT. This privilege is allocated by default to the TSOS user ID after first startup. However, when SECOS is being used, the privilege can be assigned to any desired user ID; this user ID will then have the sole authority to issue these commands for execution (see also the “SECOS” manual).

The following table contains an overview of all DSSM commands and the privileges required for command execution.

Command	Meaning	Required privilege			
		SM	O	STD	HM
ADD-SUBSYSTEM	Extend dynamic subsystem catalog	X			
HOLD-SUBSYSTEM	Place a subsystem in wait state	X	X		
MODIFY-SUBSYSTEM-PARAMETER	Modify subsystem parameters	X			
RELEASE-SUBSYSTEM-SPACE	Release reserved address space for subsystems	X		X	X
REMOVE-SUBSYSTEM	Remove inactive subsystem from dynamic catalog	X			
RESUME-SUBSYSTEM	Cancel wait state for subsystem	X	X		
SAVE-SUBSYSTEM-CATALOG	Save changes to dynamic subsystem catalog	X			
SET-DSSM-OPTIONS	Activate/deactivate DSSM logging function	X	X		
SHOW-DSSM-INFORMATION	Show information on DSSM	X			

SHOW-RESTART-OPTIONS	Display information on automatic restart		X		
SHOW-SUBSYSTEM-ATTRIBUTES	Request information on subsystem attributes	X		X	
SHOW-SUBSYSTEM-INFO	Request information on current subsystems configuration	X			
SHOW-SUBSYSTEM-STATUS	Request information on status of subsystems				
subsys=*NON-PRIV-CLASS-5		X	X	X	
subsys=*ALL / <subsys-name>				X	
START-SUBSYSTEM	Activate subsystem	X	X		
STOP-SUBSYSTEM	Deactivate subsystem	X	X		
UNLOCK-SUBSYSTEM	Shift subsystem from LOCKED status to NOT-CREATED status	X			

Table 3: DSSM commands and required privileges

SM: SUBSYSTEM-MANAGEMENT privilege for managing subsystems
O: OPERATING privilege for entry from operator terminals
STD: STD-PROCESSING privilege for executing user commands
HM: HARDWARE-MAINTENANCE privilege for calling hardware test programs

3.1 Purpose and functions of DSSM

The software product DSSM performs the following individual functions:

- DSSM sets up the entire subsystem configuration (such that it is capable of functioning).
- DSSM activates and deactivates subsystems in order to expand the subsystem configuration or adapt it to current requirements.
- DSSM checks the compatibility of individual subsystem versions when activating and deactivating subsystems.
- DSSM establishes the relations between subsystems and monitors them when activating subsystems or dissolves existing relations to other subsystems when deactivating subsystems.
- DSSM supports the linkage of tasks and programs to subsystems.
- DSSM supports the use of shared address space in memory pools.

All activities indicated below as being synchronous run in the task of the requester, while all asynchronous activities run in the holder task.

3.1.1 Subsystem declaration (SSC)

In order to manage the subsystems, DSSM requires information in the form of a declaration. This declaration determines which main components and satellites belong to a subsystem, how the subsystem can be activated, and how to set up interfaces to it. Thus, anyone creating a declaration must be familiar with the subsystem concerned. For this reason the necessary declarations for all DSSM-compatible subsystems are supplied in a declaration file.

Depending on the declaration, DSSM activates the subsystem and sets up interfaces and relations to other subsystems and to DSSM itself. Interfaces to a subsystem can be monitored by DSSM. In this way it is possible to deactivate the subsystem under the control of DSSM. If the interfaces are not monitored, the subsystem itself is responsible for deactivation.

The declarations are not given as operands in every DSSM call, as this would be too complicated and excessive. The declarations are specified using SSCM and are stored in the subsystem catalog, because

- these declarations are stable and rarely change
- these declarations are complex and extensive
- these declarations must frequently be accessed by DSSM
- not only internal subsystem information exists, but also cross-subsystem information.

The declarations of all subsystems belonging to a configuration (= declaration state) make up the SSMCAT subsystem catalog and are stored in a PAM file. This file is read in on startup and remains in memory until system shutdown. The ADD-SUBSYSTEM command can be used to extend the catalog during a session.

The LOAD-LOCAL-SUBSYSTEM-CATALOG command can be used to read the catalog and dynamically load it into the user address space of the calling task.

3.1.2 Activation and restart

Activation of a subsystem is controlled by the attributes that were defined in the subsystem catalog using the SSCM statement SET-SUBSYSTEM-ATTRIBUTES.

Activation is performed

- automatically during the first DSSM call in the startup routine if CREATION-TIME= *BEFORE-DSSM-LOAD or *AT-DSSM-LOAD was defined; activation must have been successfully completed (*BEFORE-DSSM-LOAD means that the subsystem had already been loaded before DSSM was first called by the startup routine. After DSSM is called, this subsystem will be managed like any other. Version exchange or version coexistence with another version defined with CREATION-TIME=*AT-DSSM-LOAD is permitted. It is the responsibility of systems support to ensure that one version of the subsystem is available at all times.)
- automatically during the second DSSM call in the startup routine if CREATION-TIME= *MANDATORY-AT-STARTUP was defined; in this case, activation must have been successfully completed.
- automatically during the second DSSM call in the startup routine if CREATION-TIME=*BEFORE-SYSTEM-READY was defined (synchronous).
- automatically after the second return to the startup routine if CREATION-TIME=*AFTER-SYSTEM-READY was defined (asynchronous).
- explicitly by means of the START-SUBSYSTEM command.
- explicitly as a local subsystem by means of the START-LOCAL-SUBSYSTEM command.
- explicitly when the $ESMCRE interface is called.
- implicitly after the first SVC call for a subsystem which was defined with CREATION-TIME=*AT-SUBSYSTEM-CALL(ON-ACTION=*STD or *ANY) or implicitly after the first ISL call for a subsystem that was defined with CREATION-TIME=*AT-SUBSYSTEM-CALL(ON-ACTION=*ISL-CALL or *ANY).

With reference to the start point for subsystems see also the SSCM statement SET-SUBSYSTEM-ATTRIBUTES.

In the same way as the other files required for DSSM initialization (e.g. SSMCAT) and the files required for activation of subsystems linked to startup, the program or object module library (OML) and the DSSM REP and NOREF files must have been created on the home pubset under the TSOS user ID, and must be available at startup.
Subsystems which are linked to startup are those whose activation points (creation points) are specified by the following values:

- *BEFORE-SYSTEM-READY
- *AFTER-SYSTEM-READY
- *BEFORE-DSSM-LOAD
- *AT-DSSM-LOAD
- *MANDATORY-AT-STARTUP

An indispensable precondition for activation is that the subsystem name is known, i.e. declared in the global or local subsystem catalog.

Activation takes place in the following steps:

1. The job is checked, in particular the dependencies on other subsystems (synchronous).

   If a subsystem has components which were declared with the attribute *INSTALLED and installed using IMON, DSSM calls IMON-GPN during the checking phase in order to find out the path names of the files.

   Depending on the availability of IMON-GPN and on the status of the installed installation unit (see the INSTALLATION-UNIT operand), DSSM acts as follows:

   - if the installation unit exists and a file name can be linked to the specified logical ID:

     DSSM will use this file name as long as the subsystem is active (until STOP-SUBSYSTEM) or until another file name is defined by means of the MODIFY-SUBSYSTEM-PARAMETER command.
   - In the following situations, DSSM accesses the DEFAULT-NAME defined in the catalog, if there is one:
     - IMON-GPN is not available
     - the INSTALLATION-UNIT does not exist in the IMON-GPN data
     - the INSTALLATION-UNIT exists in the IMON-GPN data but has no connection to a logical name (LOGICAL-ID)

   The message ESM0665 explains what is happening:

   ```
   ESM0665  'DEFAULT-NAME' USED FOR FILES OF SUBSYSTEM '(&00)'
   ```

   - The INSTALLATION-UNIT exists in the IMON-GPN data with a few appropriate path names but the queried logical name does not exist. DSSM now assumes that the file does not exist. Two cases are to be discriminated:
     - Subsystems that were defined with the *AT-DSSM-LOAD or *MANDATORY-AT-STARTUP attribute:
       During startup, a query that must be answered is issued to the operator terminal. The operator can now either specify a new, valid name for the file concerned (information file, module library or REP file with the REP-FILE-MANDATORY=*YES attribute) or stop the subsystem activation.
     - Subsystems that were defined with the *AT-CREATION-REQUEST, *AFTER-SYSTEM-READY or *BEFORE-SYSTEM-READY attribute:
       If one of the files (module library, information file or REP file with the REP-FILE-MANDATORY=*YES attribute) is missing, the subsystem is not activated and this is indicated with a message. If the message file or REP file (with the REP-FILE-MANDATORY=*NO attribute) is missing, the subsystem is activated but a warning is still output.
2. The subsystem is loaded in a holder task (asynchronous)
   Loading of the subsystem code is initiated by the holder task.
   In addition it makes its local address space (user address space) available for the subsystems that were defined with
   MEMORY-CLASS=*LOCAL-PRIVILEGED/*LOCAL-UNPRIVILEGED. The loading of subsystems with MEMORY-CLASS=*SYSTEM-GLOBAL is also initiated by the holder task; however, they are not loaded into the holder task’s user address space but into the shareable system address space.

   When TU subsystems are activated with MEMORY-CLASS=*BY-SLICE, the shareable program area is loaded into the shareable system address space, and the nonshareable program area and/or data area into the user address space of the holder task.

If a task establishes a connection to a subsystem of this type, only the data area that has already been loaded in the user address space of the holder task is copied to the same address in the private user address spaces of the connected tasks. This means that address-related references between the program area and data area are always possible. Performance is considerably enhanced by this method of address space distribution because no access is made to the program library or object module library when a task is connected to the subsystem, and external references do not have to be resolved.

If a subsystem was defined with MEMORY-CLASS=*BY-SLICE and is started for the first time, DSSM informs the BLSSERV subsystem that the copy of the data area in the private user address space can be accessed with the VSVI1 macro.
The VSVI1 macro informs the user about entries in the DBL tables. See the manual “BLSSERV” for details on the macro.

The holder task can also be used as the work task.

3. For TPR subsystems only: subsystem activation is started in the holder task if an INIT routine has been defined (asynchronous).
4. After completion of activation, the subsystem is opened for connections of authorized users (asynchronous).

Restarting a subsystem (RESUME-SUBSYSTEM) after a HOLD-SUBSYSTEM consists of steps 1, 3 and 4.
In step 1 it is not necessary to call IMON-GPN.

If the activation or restart of a subsystem is initiated explicitly via the START-SUBSYSTEM or RESUME-SUBSYSTEM command, the synchronous processing mode can be selected instead of the asynchronous mode.

3.1.3 Interface establishment and cancelation

The interface to the job entry point of a subsystem is implemented in one of the following ways:

1. implicitly and globally by linkage (this is possible only for interfaces to subsystems that have already been loaded or between subsystems that are loaded at the same time)
2. implicitly and task-specifically via
   - subsystem-specific SVC, ISL, bISL or system exit routines
   - in the case of nonprivileged subsystems, BLS interfaces (BIND macro, START-EXECUTABLE-PROGRAM, LOAD-EXECUTABLE-PROGRAM and START-<product-name>, autolink)
3. explicitly and task-specifically via internal system macros ($ESMCON and $ESMCCS).

Establishing an interface to a subsystem includes:

1. generating a subtask following a corresponding internal system call ($ESMCCS)
2. attachment to the memory pool, if required
3. If a subsystem was defined with MEMORY-CLASS=*BY-SLICE, the private area is copied into the local address space (user address space). When the subsystem is started for the first time, DSSM informs the BLSSERV subsystem that the copy in the private user address space can be accessed with the VSVI1 macro.
4. transferring the address to the task or calling the subsystem code
5. setting the subsystem-specific interface marker
6. incrementing the task-specific interface counter (except when defining the subsystem with the attribute CONNECTION-SCOPE=*FREE).

It is also possible to set up an interface for job entries which the subsystem manages itself.

A relation can be **canceled** in one of these ways:

- implicitly and task-specifically
  - by program/task termination
  - after a return from the subsystem job entry point if this was defined with the attribute MODE=*SVC/*ISL and CONNECTION-SCOPE=*CALL
  - after deactivation of subsystems with CONNECTION-SCOPE=*OPTIMAL
- explicitly and task-specifically via an internal system macro on termination of an SVC routine ($ESMDCN)
- implicitly and subsystem-specifically on deactivation of a subsystem which was defined with the attribute CONNECTION-SCOPE=*FREE.

Canceling a relation includes:

- detachment from the memory pool
- If the subsystem was defined with MEMORY-CLASS=*BY-SLICE, the part in the user address space of the connected task is unloaded.
  When the last connection is shut down, DSSM informs the BLSSERV subsystem that this private part can no longer be accessed.
- resetting the task-specific interface marker
- decrementing the subsystem-specific interface counter (except for subsystems defined with the attribute CONNECTION-SCOPE=*FREE)

- the “detach” function in the case of an explicit call.

The address supplied to the task at the time of interface setup can no longer be used.

3.1.4 Subsystem Deactivation or suspension

A precondition for this function is that the subsystem is active.

A subsystem can be **deactivated** in the following ways:

1. explicitly when the STOP-SUBSYSTEM command is entered (HOLD-SUBSYSTEM suspends the subsystem)
2. explicitly when privileged macros are called at the program interface ($ESMDEL and $ESMHLD)
3. implicitly when a START-SUBSYSTEM command is entered with the operand VERSION-PARALLELISM=*EXCHANGE-MODE
4. automatically at shutdown, for all subsystems whose definitions include such a declaration (STOP-AT-SHUTDOWN=*YES)

Deactivation of a subsystem (STOP-SUBSYSTEM) takes place in the following steps:

1. The job is checked, in particular the dependencies on other subsystems (synchronous).
2. The CLOSE-CTRL routine is started, provided one is defined (asynchronous).
3. The subsystem is closed for new users (asynchronous), preventing the connection of any further tasks to the subsystem (except for those with entry points with CONNECTION-SCOPE=*FREE or SVC/ISL calls for subsystems with entry points with CREATION-TIME=*AT-SUBSYSTEM-CALL).
   From this moment on, it is no longer possible to access code that has an entry point defined with CONNECTION-SCOPE=*OPTIMAL.
4. The job termination routine (STOPCOM routine), if defined, is started (asynchronous).
5. Wait until the subsystem is jobless, i.e. the subsystem-specific connection counter is 0 and no task is accessing code that has an entry point defined with CONNECTION-SCOPE=*OPTIMAL (asynchronous). Exception: if the operand FORCED=*YES is specified in the DSSM command, deinitialization is started immediately.
6. Deinitialization, if defined, is started (asynchronous).
7. Unloading from the holder task (asynchronous).

A subsystem **hold** (HOLD-SUBSYSTEM) consists of steps 1 through 6.
If the deactivation or suspension of a subsystem is initiated explicitly via the STOP-SUBSYSTEM or HOLD-SUBSYSTEM command, the synchronous processing mode can be selected instead of the asynchronous mode.

3.1.5 Swapping subsystem versions

It is possible to swap versions of a subsystem in the following three ways:

1. Deactivate the old version (STOP-SUBSYSTEM) and activate the new version of the subsystem (START-SUBSYSTEM). Under certain circumstances, this may result in very long subsystem downtimes because the new version is not activated until the old one has been completely deactivated, i.e. when all tasks have cleared their links.
2. Activate the new version of the subsystem with START-SUBSYSTEM ...,VERSION-PARALLELISM=*EXCHANGE-MODE. From this time onward, no new links to the old version of the subsystem are set up. While the last tasks are clearing their links to the old version of the subsystem, the new version is initialized. This method substantially shortens the period of subsystem unavailability.

   Provided they have been defined, the following routines are called one after the other:

   - the STOPCOM routine of the old version,
   - the INIT routine of the new version and
   - the DEINIT routine of the old version.

   For some subsystem it is potentially a problem that the DEINIT routine of the old version is running while the new version has already been activated and is running. A subsystem that was defined with VERSION-EXCHANGE=*FORBIDDEN cannot be swapped in as a new version. It can, however, be deactivated (as the old version) in exchange for a new version that has been defined with VERSION-EXCHANGE= *ALLOWED.

   The old version remains in the IN-DELETE state until there is no further task connected. If the new version is in the CREATED state, activation of the old subsystem with RESET=*YES is only possible if coexistence was approved for both versions at the time of definition.

   Activation of the old version with RESET=*YES is allowed if the new version is in the IN-DELETE state and the old version was not defined with VERSION-EXCHANGE= *FORBIDDEN.
3. The CLOSE-CTRL routine can be used to switch versions without interrupting subsystem availability.
   Provided they have been defined, the following routines are called one after the other:

   - the CLOSE-CTRL routine of the old version,
   - the INIT routine of the new version,
   - the STOPCOM routine of the old version and
   - the DEINIT routine of the old version.

   If the initialization routine of the new version does not run correctly, the old version is automatically reactivated and is in the CREATED state (the result of the CLOSE-CTRL routine is reversible), thus avoiding any interruption of subsystem availability.

   Version swapping is allowed if a version of the subsystem is in the CREATED state and all other versions of the subsystem that are declared in the catalog are in the NOT-CREATED or LOCKED state.
   A version exchange will not be executed if all declared versions of the subsystem are in the NOT-CREATED or LOCKED state. In this case the version that is activated is the one that was specified in the START-SUBSYSTEM command.

   Example

   Subsystem versions SPOOL V04.2A and V04.3A are defined with VERSION-EXCHANGE=*ALLOWED.

```
/SHOW-SUBSYSTEM-STATUS SPOOL,*ALL
SUBSYSTEM SPOOL    /V04.2.A IS NOT CREATED
SUBSYSTEM SPOOL    /V04.3.A IS NOT CREATED
/START-SUBSYSTEM SPOOL,04.2.A,VERSION-PARAL=*EXCHANGE-MODE,SYNCH=*YES
ESM0220  FUNCTION 'CREATE' FOR SUBSYSTEM 'SPOOL /V04.3.A' COMPLETELY
         PROCESSED
ESM0400  'CREATE' OR 'RESUME' SUBSYSTEM 'SPOOL /V04.3.A' WITH
         'SYNCHRONOUS=YES' AND 'RESET=NO'
ESM0220  FUNCTION 'CREATE' FOR SUBSYSTEM 'SPOOL /V04.3.A' COMPLETELY
         PROCESSED
```

If there are one or more versions of the subsystem which are not in the NOT-CREATED or LOCKED state (apart from the version in the CREATED state that is to be
exchanged), the exchange will be rejected, even if all versions allow coexistence.

Example

Subsystem versions UTM V06.3, V06.4 and V06.5 are defined with
VERSION-COEXISTENCE=*ALLOWED and VERSION-EXCHANGE=*ALLOWED.

```
/SHOW-SUBSYSTEM-STATUS UTM,*ALL
SUBSYSTEM UTM    /V06.5    IS NOT RESUMED
SUBSYSTEM UTM    /V06.5    IS NOT CREATED
SUBSYSTEM UTM    /V06.5    IS CREATED
/START-SUBSYSTEM UTM,06.4,VERSION-PARALLELISM=*EXCHANGE-MODE
ESM0206  SOME ACTIONS IN PROGRESS FOR SUBSYSTEM 'UTM/V06.3'.
         NO FURTHER ACTION ON ANOTHER VERSION POSSIBLE
ESM0224  REQUESTED FUNCTION 'CREATE' FOR SUBSYSTEM 'UTM/V06.4'
         REJECTED
```

When the old version of the subsystem is replaced with a new one, the syntax file of the new version is also loaded. This means that the syntax of the new version must also recognize and execute commands and statements of the old version, i.e. it must be ensured that the syntax of the new version supports that of the old version.

It is advisable to use the CLOSE-CTRL routine to swap versions only when the new version which is to be activated is also the higher of the two.

3.1.6 Coexistence of versions

DSSM has offered the option of keeping two versions of a subsystem active, temporarily or permanently, to permit all the tasks which are linked to the old version to continue working. This coexistence mode must be specified when the subsystem is defined (using the SSCM statement SET-SUBSYSTEM-ATTRIBUTES ...,VERSION-COEXISTENCE=*ALLOWED), and must be explicitly requested in the START-SUBSYSTEM command.

If a **temporary coexistence** is in effect and version B of a subsystem is loaded whilst version A of the subsystem is active, all new callers will be connected to version B. Jobs which are connected to version A will still be processed. When all the jobs which use version A have been processed, this version will automatically be terminated.
It should be noted that, in the definition, the “old” version which is being replaced must not be dependent on the “new” version which replaces it.

If a **permanent coexistence** of different versions of a subsystem is required, then the appropriate attribute must be specified in the definition for each of the subsystems, and this mode must be explicitly requested in the START-SUBSYSTEM command.

3.1.7 Relations between subsystems

DSSM acts as the central mechanism responsible for managing the interrelations between the subsystems, allowing various relations to be explicitly monitored.

Job relations to a subsystem via explicit or implicit connections are set up and monitored by DSSM. A job relation can always be established between a task and a job entry.

Address relations via linkage editors and loaders must be specified explicitly in an operand of the SSCM statement SET-SUBSYSTEM-ATTRIBUTES (REFERENCED-SUBSYSTEMS). However, the operand should only be applied to subsystems from the same “family” (privilege level, memory class, etc.). In the case of unbundled subsystems, system administration should only select dynamic addressing mechanisms (SVC, ISL or system exit mechanisms for privileged subsystems; the BLS interface, BIND, for nonprivileged subsystems).

Address relations restrict the unloadability of a subsystem, since unloading has to take place in the reverse order from loading. Address relations are taken into account during loading and unloading, but they are not monitored. They prevent a subsystem from being unloaded, regardless of whether or not jobs exist.

Dependency relations require the availability of another subsystem. These relations can be declared by means of the RELATED-SUBSYSTEMS operand of the SSCM statement SET-SUBSYSTEM-ATTRIBUTES and are taken into account in the loading and unloading sequences.

3.1.8 Relations between subsystem satellites and subsystems

During the installation of a subsystem, the references to its satellites are created.

The **module library** is a mandatory satellite, as no subsystem can be loaded without it.

If an **information file** or a **syntax file** is specified in the definition, they will also be given the status of mandatory components, which will prevent the subsystem from being loaded if they are missing.

If a **REP file** is specified in the definition, this will only be mandatory at load time if this has been explicitly stated in the definition (REP-FILE-MANDATORY=*YES).

A **message file** can be specified in the definition, but this is not a mandatory component. In other words, loading of the subsystem can be carried out even if the message file is missing.

In following situations, DSSM sends a query to the operator terminal that must be answered by the operator:

- the module library of a mandatory subsystem is missing (*AT-DSSM-LOAD or *MANDATORY-AT-STARTUP attribute)
- the specified information is declared but not present
- the REP file with the REP-FILE-MANDATORY=*YES attribute is missing

Despite a missing file, the operator still has the option of continuing the startup process (e.g. starting the subsystem without the information file). The operator is exclusively responsible for a possible abnormal subsystem or system termination.

3.1.9 Communication between subsystems and DSSM

The exchange of information and messages is essential for subsystem-specific routines for initialization, deinitialization, job termination and the check on job termination (INIT, DEINIT, STOPCOM and CLOSE-CTRL routines).

The communications area always consists of an information area for the started routine (DSSM --> subsystem) and an acknowledgment area (subsystem --> DSSM).

The routine is started via

1. procedure call
   during initialization (with no condition) or at deinitialization, job termination and the check on job termination if the holder task is not being used as a work task
2. the interface (bourse or FITC) transferred during initialization
   if the holder task is being used as a work task

Acknowledgment to DSSM is given via a procedure return with an acknowledgment area at deinitialization, job termination and the check on job termination if the holder task is not or is no longer being used as a work task. If the holder task is being used as a work task, DSSM is notified via a NOTIFY call with the acknowledgment area serving as an input parameter.

3.1.10 Information about subsystems

Users can request information about the status of a subsystem by means of the SHOW-SUBSYSTEM-STATUS and SHOW-SUBSYSTEM-INFO commands. They can also obtain an overview of the overall subsystem configuration.

3.1.11 Status of a subsystem

NOT-CREATED

The subsystem has been declared in the current system, but has not yet been activated by a START-SUBSYSTEM command, or has been deactivated again since being activated. Tasks cannot access this subsystem until it has been activated.

IN-CREATE

The subsystem is being activated, but loading and initialization have not been completed. Tasks cannot access this subsystem yet.

CREATED

The subsystem has been loaded and initialized. Tasks can access the subsystem.

IN-DELETE

The subsystem has been deactivated by a STOP-SUBSYSTEM command. Unloading and deinitialization have not yet been completed.
Other tasks can no longer access this subsystem. Processing of tasks still connected to the subsystem will be completed.

IN-HOLD

The subsystem has been suspended by a HOLD-SUBSYSTEM command. Deinitialization has not yet been completed.
Other tasks can no longer access this subsystem. Processing of tasks still connected to the subsystem will be completed.

IN-RESUME

The subsystem is being resumed by a RESUME-SUBSYSTEM command. Reinitialization has not yet been completed. Tasks cannot yet access this subsystem.

NOT-RESUMED

The subsystem has been suspended by a HOLD-SUBSYSTEM command. Deinitialization has been completed. Tasks cannot access this subsystem until a RESUME-SUBSYSTEM command has been issued and successfully executed.

LOCKED

An unrecoverable error has occurred while the subsystem was active or was being activated, deactivated, resumed or suspended. Any further attempt to issue the corresponding commands will be rejected.

The following situations can put a subsystem into the LOCKED status:

- if the change to this status is specified in the INIT, DEINIT, STOPCOM or CLOSE-CTRL routine (applies only to privileged subsystems);
- if the subsystem’s holder task has terminated abnormally and either no restart is provided for this task, or it cannot be executed (see "Error handling in DSSM") or
- if a problem occurs in deactivating the old version during a version swap that entails an interruption of subsystem availability; regardless of the value of the RESTART operand, the subsystem is in the LOCKED status; activation of the new version is continued.

When a subsystem is activated, deactivated, suspended or resumed, its status changes, i.e. it progresses from an initial status to a final status (for example, when a subsystem is activated, its final status is CREATED).
The initial status for a request may not always be the same; e.g. a START-SUBSYSTEM command is acceptable for a subsystem with the status NOT-CREATED or IN-DELETE if the parameter RESET=*YES has been set.

The different statuses possible for a subsystem are summarized in the following table. The changes of status caused by a DSSM command are shown on a single line. The initial status of the subsystem is indicatesd by a 1. The highest digit in the line denotes the final status attainable by the appropriate DSSM command. Possible intermediate statuses are also shown.

DSSM commands	**States of a subsystem**							**Operand of the DSSM command**[1]
	NOT-CREATED	IN-CREATE	CREATED	IN-DELETE	IN-HOLD	IN-RESUME	NOT-RESUMED	
START-SUBSYSTEM	1	2	3					---
			3			2	1	RESET=*YES
	1	2	3					RESET=*YES
		2	3	1				RESET=*YES
			3		1	2		RESET=*YES
			3			2	1	RESET=*YES
STOP-SUBSYSTEM	3		1	2				---
	3			2			1	---
	2				1			FORCED=*YES
	2			1				FORCED=*YES
HOLD-SUBSYSTEM			1		2		3	---
					1		2	FORCED=*YES
RESUME-SUBSYSTEM			3			2	1	---
			3			2	1	RESET=*YES
			3		1	2		RESET=*YES

[1]Operand of the DSSM command which must be set in order to change t he status

Table 4: Statuses of a subsystem

3.1.12 Subsystem monitoring with monitoring job variables

Subsystems can be monitored with monitoring job variables (MONJV). The MONJV must be specified in the START-SUBSYSTEM command. DSSM administers and sets the MONJV during the entire subsystem runtime until it is shut down, with:

- explicit DSSM calls (/HOLD-SUBSYSTEM, /RESUME-SUBSYSTEM, /STOP-SUBSYSTEM, /UNLOCK-SUBSYSTEM)
- implicit operations (subsystem, automatic restore ...)
- SHUTDOWN

The MONJV indicates whether the subsystem is active, stopped, interrupted or locked. The MONJV can have the following contents:

Byte	Length	Contents	Values
1	3	Status	$R (running) / $A (abnormal end) / $L (loaded) / $T (terminate)
4	1	Reserved	0
5	4	TSN	???? (*four question marks*)
9	4	Home Catid	
13	4	Reserved	
17	3	Session number	
18	51	Time stamp	
71	3	Session number	
74	8	Name of the subsystem	
82	7	Version of the subsystem	
89	15	Condition of the subsystem	for $R: created for $A: abnormal end / locked for $L: in create for $T: not created / not resumed / in delete / in resume / in hold
104	24	Unused	
128	127	Reserved for subsystem users	

Table 5: Contents of the monitor job variables

3.1.13 Overview of functions

This table provides an overview of the functions offered by DSSM/SSCM, and shows which operands may be specified for the individual subsystem classes.

Function (operands)	**TPR SAR**	**TU SAR**	**TU BAR**	**Sys-exits**	**Share Prod.**	**PU**
Subsystem declaration (SSCM) statement SET-SUBSYSTEM-ATTRIBUTES						
Identification						
SUBSYSTEM-NAME, VERSION,-DYNAMIC-CHECK-ENTRY	X	X	X	X	X	
Linking and loading						
LIBRARY, REP-FILE, LINK-ENTRY, AUTOLINK,-UNRESOLVED-EXTERNALS	X	X	X	X	X	
CREATION-TIME=*BEFORE-DSSM-LOAD/ *AT-DSSM-LOAD/*BEFORE-SYSTEM-READY	X			X		
CREATION-TIME=*AFTER-SYSTEM-READY/ *AT-CREATION-REQUEST	X	X	X	X	X	
CREATION-TIME=*AT-SUBSYSTEM-CALL	1					
REFERENCED-SUBSYSTEMS	2	2	2	2	2	
Address space						
MEMORY-CLASS=*SYSTEM-GLOBAL	X	X		X	10	
MEMORY-CLASS=*=*LOCAL-PRIVILEGED			X		10	
MEMORY-CLASS=*=*LOCAL-UNPRIVILEGED			X		10	
MEMORY-CLASS=*=*BY-SLICE		X	X		10	
SUBSYSTEM-ACCESS=*SYSTEM	X			X		
SUBSYSTEM-ACCESS=*=*HIGH		X	X		X	
SIZE			X		X	
START-ADDRESS			3		3	
Subsystem satellites						
MESSAGE-FILE, SYNTAX-FILE	X	X	X	8	8	
SUBSYSTEM-INFO-FILE	X			8		
Starting and terminating						

INIT-, STOPCOM-, DEINIT-, CLOSE-CTRL-routine,-INTERFACE-VERSION	X			X		
Holder task for execution						
RESTART-REQUIRED	X			X		
Execution (operand SUBSYSTEM-ENTRIES=(...))						
MODE=*LINK	X	X	X		X	
MODE=*SVC/*SYS-EXIT/*ISL	X			X		
CONNECTION-ACCESS=*SYSTEM	X			X		
CONNECTION-ACCESS=*ALL		X	X		X	
CONNECTION-SCOPE=*PROGRAM/*TASK/*FREE	X	X	X	X	X	
CONNECTION-SCOPE=*CALL/*OPTIMAL	9					
Subsystem configuration (SSCM) statements						
ASSIGN-HOLDER-TASK	X	4	X		8	
SET-SUBSYSTEM-ATTRIBUTES (Operand RELATED-SUBSYSTEMS)	X	X	X	X	X	
SET-SUBSYSTEM-ATTRIBUTES (Operand REFERENCED-SUBSYSTEMS)	11	11	11	11	11	
SEPARATE-ADDRESS-SPACE			5		5	
Subsystem management (DSSM) commands						
START-SUBSYSTEM, STOP-SUBSYSTEM,-HOLD-SUBSYSTEM, RESUME-SUBSYSTEM	X	X	X	X	X	
ADD-/REMOVE-/UNLOCK-SUBSYSTEM,-SAVE-SUBSYSTEM-CATALOG						X
MODIFY-SUBSYSTEM-PARAMETER	X	X	X	X	X	X
Subsystem information (DSSM) commands						
SHOW-SUBSYSTEM-ATTRIBUTES						X
SHOW-SUBSYSTEM-INFO						X
SHOW-SUBSYSTEM-STATUS	X	X	X	X	X	
Global subsystem management (DSSM) commands						
RELEASE-SUBSYSTEM-SPACE			6			
SET-DSSM-OPTIONS	7	7	7	7	7	7

Table 6: Overview of DSSM functions

Key

TPR SAR	Privileged subsystems (only system address space)
TU SAR	Nonprivileged subsystems with system address space
TU BAR	Nonprivileged subsystems with user address space
Sys. exits	Relevant for system exits
Share prod.	Relevant for share products
PK	Privileged user ID for systems support (user ID with SUBSYSTEM-MANAGEMENT privilege)
X	Function available
1	Only for subsystems with SVC and/or ISL connection
2	Relations only from UAS to SAS
3	only for subsystems in class 6 memory
4	Holder task is required in system address space only for execution purposes
5	Required only for subsystems in the address space strip
6	Reservation for the nonprivileged address space strip or both strips can be canceled
7	Only for diagnosis and debugging
8	Only where appropriate
9	Only in conjunction with MODE=*SVC or MODE=*ISL
10	Depending on storage location in the address space
11	Should be reserved for subsystems in the same “family” (privileges, memory class, etc.)

3.2 Storage and task concepts

Storage concept

Subsystems are loaded as follows, depending on the declaration:

- into the system address space (class 3 or class 4 memory)
  if MEMORY-CLASS=*SYSTEM-GLOBAL
- into the user address space (memory pool: class 5 or class 6 memory)
  if MEMORY-CLASS=*LOCAL-PRIVILEGED/*LOCAL-UNPRIVILEGED
- into both address spaces if MEMORY-CLASS=*BY-SLICE

Interfaces to subsystems in the user address space are possible only if the allocated address area in the local task is free. The code must always be located at the same address after linkage. For generally accessible subsystems in the user address space, which can be addressed by any task at any time, a fixed area in class 5 memory is reserved, known as the “address space strip”. This address space strip is shared by all generally available subsystems, with a separate strip being available for each nonprivileged subsystem.

Problems can occur for user tasks if there is not enough class 5 memory space. The address space strip reserved for subsystems can be released by means of the DSSM command RELEASE-SUBSYSTEM-SPACE in order to obtain more class 5 memory space.

Subsystems that are not used at the same time and therefore do not call each other may be located in parallel within the address space. The more parallel subsystems there are, the smaller is the address space strip required. Care should be taken to ensure the right balance of parallel subsystems, since a high degree of parallelism, though it saves address space, may result in a deterioration of performance (the more parallelism, the more mutual preemption). The distribution of the subsystems within the address space strip can be controlled using the SSCM statement SEPARATE-ADDRESS-SPACE.

Address space housekeeping

DSSM address space housekeeping provides a means of reducing the load on the system address space (class 3 and class 4 memory) by putting subsystems in the address space of the holder task. This is only of relevance to nonprivileged subsystems, however, because all privileged subsystems are always loaded into the system address space (MEMORY-CLASS=*SYSTEM-GLOBAL).
Transfer is straightforward, provided that the subsystems do not call each other, do not depend on each other through a third program, and do not have the ability to run in class 6 memory as a main program.
Note that it is only possible to attempt to relocate nonprivileged subsystems which are currently below the 16-Mbyte boundary.
For all other nonprivileged subsystems, loading above the 16-Mbyte boundary is recommended (SUBSYSTEM-ACCESS=*HIGH).

Subsystems which are reentrant-compatible and run as main programs can also be swapped out of the system address space. They must be available in the memory pool in the class 6 memory (MEMORY-CLASS=*LOCAL-UNPRIVILEGED).

There is a strategy aimed at minimizing system address space occupancy for subsystems that consist of a shareable code (program area) and a non-shareable code (data area). This strategy works as follows:
The program area is loaded into the shareable address space (this corresponds to MEMORY-CLASS=*SYSTEM-GLOBAL). The data area is loaded into the user address space of the holder task; then, when a task is linked to the subsystem in the task's private user address space, it is copied to the same address.
This strategy is implemented when a subsystem is defined with MEMORY-CLASS=*BY-SLICE.

The advantages of this approach are as follows:

- address-related references between the program area and data area are always possible because the copy of the data area begins at the same address as the original
- performance is considerably enhanced by this type of address space apportionment because no access is made to the program library or object module library when a task is linked to the subsystem, and external references do not need to be resolved.

The disadvantages of this approach are:

- once the address space for the data area has been defined and reserved at the time of startup, it can be expanded to only a very limited degree; optimization of memory space is not possible because of the strict apportionment of address space
- if the address space belonging to the task being linked and intended for the data area is already occupied, the subsystem code (data area and program area) is loaded in its entirety into the user address space of that task.

Relocation from the system address space to the user address space is only worthwhile in the following cases:

1. Where subsystems which are independent of each other and are not used simultaneously by user programs can be placed in parallel in the user address area. If this is not the case, relocation is pointless, since they will take up more memory in the user address space than in the system address space.
2. Where the subsystems are large enough to compensate for the loss that occurs through the use of the memory pool (minimum size of a memory pool: 1Mbyte).
3. Where the subsystem must be located below the 16-Mbyte boundary (for subsystems which may also be loaded above this boundary, the operand SUBSYSTEM-ACCESS= *HIGH can be used to counteract any overloading of the lower 16 Mbytes).

Summary:

It is only appropriate to use parallel configuration of independent subsystems in memory pools if the sum of the sizes of all the subsystems amounts to over 1Mbyte and none of the subsystems is larger than 1Mbyte in size.

Task concept - holder task

A subsystem is activated under its own task, the holder task. Depending on the type of subsystem, this task can be used as a subsystem work task or as a pure holder task. The user address space of this task can be used for relocation from the system address space.

The number of holder tasks required should be kept as small as possible. A large number of tasks does have a positive effect on parallelism, since the more tasks are created at the same time, the more subsystems can be installed. On the other hand, more tasks also require more task-specific tables.

DSSM itself minimizes the number of holder tasks. However, the distribution of subsystems can also be controlled with SSCM by means of the ASSIGN-HOLDER-TASK statement (see "ASSIGN-HOLDER-TASK") during generation of the subsystem catalog.
The default option is that all subsystems defined with MEMORY-CLASS=*BY-SLICE are connected to the same holder task.

If an error occurs, a restart of the holder task is automatically initiated, to avoid bringing down all the subsystems which are connected to the holder task. Provision is also made for restarting a subsystem by means of the RESTART-REQUIRED operand of the SET-SUBSYSTEM-ATTRIBUTES command. This parameter makes it possible to call the subsystem initialization again if the holder task was terminated during the execution of subsystem routines (see "Error handling in DSSM").

3.3 Management of shared programs

DSSM supports the management of shared programs. This permits the following during a session:

- unloading of shared programs, even from class 4 memory
- relocation of shared programs from class 4 memory to class 5 or class 6 memory.

Shared programs can be declared as subsystems and as such administered by DSSM.They can be dynamically activated, deactivated, suspended and resumed, just like any other subsystems.

This requires that the shared programs be declared during generation of the subsystem catalog. Given this definition, a shared program can be activated and deactivated like a normal subsystem after DSSM has been started.

Users can access the program via the BIND macro or by means of the START-EXECUTABLE-PROGRAM and LOAD-EXECUTABLE-PROGRAM commands.

The entry point must always be defined by means of the SSCM statement SET-SUBSYSTEM-ATTRIBUTES,...SUBSYSTEM-ENTRIES=, even if it is identical with the LINK-ENTRY (of the reference address used to load the subsystem).

3.4 Management of the dynamic subsystem catalog

Adding to the dynamic subsystem catalog

System administration can extend the current dynamic subsystem configuration during a session (ADD-SUBSYSTEM command). The catalog for the new subsystem configuration must be generated using SSCM, and the following points must be noted:

- If required, the new catalog can be larger that the old one (i.e. it may also include all the subsystems from the old catalog), or it may be created as a “delta” catalog, containing only the new subsystem definitions.
- The “old” subsystem catalog, which was used during startup, will not automatically be updated. At the next startup it is possible
  - to use the “new” catalog, or
  - to use instead a catalog which has been completely recreated and which, for example, does not contain the versions of subsystems which are no longer being used, and in which the required changes have been made to attributes (such as CREATION-TIME).
- ASSIGN-HOLDER-TASK statement

  This statement must not be specified for old and new subsystems.

  Example

Subsystems in the old catalog: Subsystems in the new catalog:	A, B, C A, B, C, D and E

Then the following is allowed:
`//ASSIGN-HOLDER-TASK TYPE=SHARED-HOLDER(BY-SUBSYSTEMS=(A,B))`
`//ASSIGN-HOLDER-TASK TYPE=SHARED-HOLDER(BY-SUBSYSTEMS=(D,E))`	

and the following is *not* allowed: `//ASSIGN-HOLDER-TASK TYPE=SHARED-HOLDER(BY-SUBSYSTEMS=(A,B,E))`	

- SET-SUBSYSTEM-ATTRIBUTES statement

  The CREATION-TIME operand for a new subsystem must be compatible with any versions of the subsystem which already exist in the old subsystem configuration.

  Example

  If version 1 of subsystem X in the old catalog is declared with CREATION-TIME=*AT-SUBSYSTEM-CALL, then version 2 of subsystem X in the new catalog must have CREATION-TIME=*AT-CREATION-REQUEST.

  If CREATION-TIME=*BEFORE-SYSTEM-READY or *AFTER-SYSTEM-READY has been specified for a new subsystem, the subsystem will not be created because the SYSTEM READY point in time (i.e. system initialization) will not occur. A corresponding message will be issued.

  New subsystems which are being added and have the attribute MEMORY-CLASS=*LOCAL-PRIVILEGED (class 5 memory) must not alter the width of the system or user address space strip, nor the location of the old subsystems within this strip.

  No relations may be declared from the old to the new subsystems (REFERENCED-SUBSYSTEM and RELATED-SUBSYSTEM operands).

  Example

Subsystems in the old catalog: Subsystems in the new catalog:		A, B, C A, B, C, D and E
X --> Y means that X requires Y in order to resolve external calls.		
Then:	A --> D D --> E D --> B,C	not allowed is allowed is allowed.

- SEPARATE-ADDRESS-SPACE statement

  New and old subsystems may be disjunctive, i.e. their addresses may overlap.

  Example

Subsystems in the old catalog: Subsystems in the new catalog:		A, B, C A, B, C, D and E
X --> Y means that X is disjunctive to Y.		
Then:	A --> D D --> E D --> B,C	is allowed is allowed is allowed.

Changing subsystem attributes during a session

The MODIFY-SUBSYSTEM-PARAMETER command can be used to change subsystem attributes during a session, without the user having to shut down the subsystem configuration or add a new version of the subsystem. Changes in the configuration can be saved for the next startup by means of the SAVE-SUBSYSTEM-CATALOG command.

These possibilities are helpful especially in the following situations:

- In order to stop a subsystem, all the connected tasks must have cleared their links. If the subsystem was not defined with the attribute FORCED-STATE-CHANGE=*ALLOWED, there is no way of accelerating the deactivation of the subsystem. Should certain considerations make it necessary to do so, however, it is now possible to change the attribute dynamically by means of the MODIFY-SUBSYSTEM-PARAMETER command.
- If memory pool contention occurs because, for example, two subsystems are located at the same address and the task can use only one of the two, the addresses of the subsystems can be changed to avoid memory pool contention by means of the MODIFY-SUBSYSTEM-PARAMETER command.

3.5 Startup of dynamic subsystem management

Dynamic subsystem management is started during BS2000 system initialization.

All the information necessary for DSSM initialization is entered via the parameter service. This information includes the name of the static subsystem catalog and the DSSM version number. If absolutely necessary, logging of DSSM-specific data for error diagnosis may be activated at this point.

All the records processed by means of the parameter service are listed in the form of messages in the CONSLOG logging file.

The keyword for starting up subsystem management is **DSSM**.

The procedure for starting up DSSM via the parameter service is described in detail in themanual “Introductory Guide to Systems Support”.

In order for DSSM V21.0 to operate, the following files must be stored on the home pubset under the TSOS user ID:

SYSLNK.DSSM.210	Library with load modules for DSSM
SKMLNK.DSSM.210	Library with load modules for DSSM x86 variant
SYSNRF.DSSM.210	Reference file for DSSM REP file processing (NOREF file)
SYSREP.DSSM.210	REP correction file for DSSM
SYSSDF.DSSM.210	SDF syntax file
SYSMES.DSSM.210	Message file

The subsystem catalog which is to be created must likewise be placed on the home pubset and stored under the TSOS user ID. The catalog may be named as desired, and the name can be made known to the system via the parameter service.

The SSCM program is available for generating a subsystem catalog. This program is described in detail in section "SSCM".

Problem handling during a system run

If DSSM cannot be initialized, the reason is displayed in a message (e.g. missing static subsystem catalog) and message `ESM0401` is output. The operator can specify a new subsystem catalog during the system run on the operator terminal if it was not defined in the parameter file or the standard catalog (SYS.SSD.CAT.X) is not present.

The system run is generally not continued if mandatory subsystems cannot be started up. The reason for the error during DSSM initialization must first be eliminated and the system run must then be repeated.

If subsystems with the *BEFORE-SYSTEM-READY attribute cannot be started up, DSSM continues the system run.

If one of the following files of a subsystem with the *AT-DSSM-LOAD or *MANDATORY-AT-STARTUP attribute is missing, the operator can specify a new, valid name during the system run:

- information file
- REP correction file
- module library

The system run is stopped if the operator does not input a new file name for the missing REP correction file or the module library.

If the information file is missing, the operator can

- input a new, valid file name
- ignore the message and continue the system run
- ignore the message and stop the system run.

Format of the parameter record for starting up DSSM

Format	Meaning
SSMCAT=name	Name of the static subsystem catalog
VERSION=versno	Version number of DSSM
LOGGING=ON / OFF	Controls DSSM-specific logging for error diagnostics
REPFILE=<repfile name>	Name of the REP correction file for loading DSSM

START-SUBSYSTEM commands must be issued in the BS2000 session for any subsystems that are not set up automatically during system initialization.

Excerpt from the parameter file

```
/BS2000 PARAMS
:
/BEGIN DSSM
 SSMCAT=name ————————————————————————————————————————————  (1)
 VERSION=versno ——————————————————————————————————————————  (2)
 LOGGING=ON / OFF ————————————————————————————————————————  (3)
 REPFILE=repfile name————————————————————————————————————  (4)
/EOF
:
/END-PARAMS
```

(1) Control and parameter records must exist in the parameter file only if the system support wishes to set values different from the following default values:

for BS2000/OSD-BC V5.0: SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=040

for BS2000/OSD-BC V4.0: SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=040
or
SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=039
or
SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=038

for BS2000/OSD-BC V3.0: SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=040
or
SSMCAT=$TSOS.SYS.SSD.CAT.X and VERSION=036

(2) The version number refers to all DSSM-specific file names(e.g. SYSLNK.DSSM.040, SYSREP.DSSM.040 in the case of BS2000/OSD-BC V5.0).

(3) The statement LOGGING=OFF deactivates logging. With LOGGING=ON a log containing diagnostics data is created during DSSM startup. Logging is not possible if only default values are specified.

(4) Name of the desired REP correction file for loading DSSM.If the parameter is not specified, DSSM is loaded with the default name of the REP correction file (SYSREP.DSSM.*version*).

3.6 DSSM accounting records

General comments on the BS2000 accounting system

The BS2000 accounting system as a whole is controlled by systems support. Systems support determines the time at which the accounting system is to be started, declares the name of the accounting file, and defines the name and number of accounting records and record extensions that are to be recorded in the accounting file.
Systems support also determines the cycle and scope of periodic data entry encompassing certain accounting records and job classes.

The accounting system can be used by systems support to activate and deactivate accounting records, either entirely or in part, during a session, and to influence the extent of the individual accounting records.
Accounting records and record extensions which are not required can be deactivated by means of the
MODIFY-ACCOUNTING-PARAMETERS command.

The attributes used in the representation of each data field are as follows:

Field	Consecutive number of the data field within the described part of the record
Displacement	Relative distance of the data field from the start of the described part of the record
Length	Length of the data field in bytes
Format	A = alphanumeric B = binary number B2 =binary representation of CPU time C = printable characters, including special characters F = file name X = non-printable characters Z = unpacked decimal number (*) * = specified for the individual record types or extension elements - = reserved for future extensions; contains either a space or binary zero
(*)	Time is shown in the form `hhmmss`, the date in the form `yymmdd`

An accounting record consists of the following four parts:

(A) Record description	record identifier, time stamp, ...
(B) Identification section	user ID, account number, monitored resources , ...
(C) Basic information	default data
(D) Variable information	record extensions

For more details about the BS2000 accounting system refer to the manual “Introductory Guide to Systems Support”.

Subsystem record description

The record description (A) contains a record identifier, which serves the purpose of differentiating the individual accounting records, a time stamp, and details of the length of the identification section and of the basic information.

Address of the record description = left-hand end of record

Structure and contents:

Field no.	Displacement		Length	Format	Meaning
	hex.	dec.	(bytes)		
1	00	0	4	A	Record identifier [1]
2	04	4	8	B	Time stamp of time of day clock [2]
3	0C	12	2	B	Length of identification section
4	0E	14	2	B	Length of basic information
5	10	16	4	-	- reserved -

[1] The record identifier serves to differentiate between the individual record types.

[2] The time stamp is entered in the record by the system as the last item of information after creation of the record is completed or after it has been transferred to the accounting system.

Length of the record description: 20 bytes

Subsystem identification

The subsystem identification in the identification section (B) describes the subsystem to which the subsystem account data relates.

Structure and contents:

Field no.	Displacement		Length	Format	Meaning
	hex.	dec.	(bytes)		
1	00	0	8	A	Name of the subsystem
2	08	8	7	A	Version of the subsystem
3	0F	15	8	Z	Date of the call [1]
4	17	23	6	Z	Time of the call [2]

[1] Date in the form `yyyymmdd`

[2] Time in the form `hhmmss`

Length of the DSSM identification section: 29 bytes

3.6.1 ESMC - subsystem initialization accounting record

The accounting record is written every time that the initialization phase of a subsystem is executed. This activation routine traverses the subsystem under the control of DSSM on execution of the START-SUBSYSTEM and RESUME-SUBSYSTEM commands.

Maximum length of the subsystem initialization accounting record: 54 bytes

(A) Record description: record identifier: “ESMC”

(B) Identification section: subsystem identifier

(C) Basic information

Field no.	Displacement		Length	Format	Meaning
	hex.	dec.	(bytes)		
1	00	0	1	B	Status indicator [1]
2	01	1	1	A	Season identifier (current) [2]
3	02	2	1	-	- reserved -

[1] There are two possible values for the status indicator:
1 subsystem is restarted after the wait state (RESUME-SUBSYSTEM command)
0 subsystem is started (START-SUBSYSTEM command)

[2] “S” for summer time; “W” for winter time

Length of the basic information: 3 bytes

(D) Variable information

The variable information of the subsystem initialization accounting record contains **no** record extension.

3.6.2 ESMD - subsystem termination accounting record

The accounting record is written every time that the deinitialization phase of a subsystem is executed. This termination routine traverses the subsystem under the control of DSSM on execution of the STOP-SUBSYSTEM and HOLD-SUBSYSTEM commands.

Maximum length of the subsystem termination accounting record: 54 bytes

(A) Record description: record identifier: “ESMD”

(B) Identification section: subsystem identification

(C) Basic information

Field no.	Displacement		Length	Format	Meaning
	hex.	dec.	(bytes)		
1	00	0	1	B	Status indicator [1]
2	01	1	1	A	Season identifier (current) [2]
3	02	2	1	-	- reserved -

[1] There are two possible values for the status indicator:
1 subsystem is placed in the wait state (HOLD-SUBSYSTEM command)
0 subsystem is terminated (STOP-SUBSYSTEM command)

[2] “S” for summer time; “W” for winter time

Length of the basic information: 3 bytes

(D) Variable information

The variable information of the subsystem termination accounting record contains **no** record extension.

3.7 Error handling in DSSM

DSSM task error

1. DSSM terminates abnormally during the first step of startup if, for example, subsystems are activated that were defined with the attribute *BEFORE-DSSM-LOAD or *AT-DSSM-LOAD. An error code is sent to the startup task.
   Normally, startup is aborted. However, under certain circumstances it may be continued without DSSM initialization (depending on how startup is implemented).

   For information on the startup steps see "MODIFY-SUBSYSTEM-ATTRIBUTES" or the manual “Introductory Guide to Systems Support”.
2. DSSM terminates abnormally during the second step of startup if, for example, subsystems are activated that were defined with the attribute *MANDATORY-AT-STARTUP or *BEFORE-SYSTEM-READY or while the data structures of a subsystem with the attribute *AFTER-SYSTEM-READY are being updated. If DSSM attempted to activate one of these subsystems, its status is changed to LOCKED.

   A return code is sent to the startup task when a subsystem with *MANDATORY-AT-STARTUP is put in the LOCKED status. Whether or not startup is aborted depends on how startup is implemented.
3. DSSM terminates abnormally during shutdown if the subsystem which is to be deactivated is in the LOCKED status.
   The shutdown of other subsystems continues normally.
4. Other causes that lead to abnormal DSSM termination:
   DSSM analyzes the situation, restores the integrity of its internal tables and makes the following decisions, depending on where the error occurred:

The error occurs during ...	**Reaction**
CREATE/DELETE/RESUME/HOLD	Subsystem LOCKED (with error message)
the swapping of subsystem versions	The faulty routine is called again; if it again results in the error,
• with interrupted availability	• the version concerned are placed in the LOCKED status,
• without interrupted availability	• both subsystem versions are placed in the LOCKED status.
ADD-/MODIFY-SUBSYSTEM-PARAMETER, REMOVE-SUBSYSTEM	The statement is aborted; parts of the catalog may already have been changed.
restoration of the holder task	Restoration is aborted.
SHOW-SUBSYSTEM-INFO, SAVE-CATALOG	The request is terminated abnormally.

   After restoring the integrity of the tables, DSSM resumes its work with the requests waiting in the DSSM bourse.

Holder task error

If problems occur in the holder task or it is terminated abnormally, DSSM analyzes the situation, automatically initiates the restart of the holder task and makes the following decisions:

The problem occurs ...	Reaction to the holder task error when	
	RESTART-REQUIRED=*YES	**RESTART-REQUIRED=*NO**
• when normal demands are being made on the subsystem		
Activation	Subsystem LOCKED	Subsystem LOCKED
INIT routine	INIT routine called	Subsystem LOCKED
CLOSE-CTRL routine	Subsystem CREATED	Subsystem LOCKED
STOPCOM routine	INIT routine called	Subsystem LOCKED
DEINIT routine	Deinitialization continued	Subsystem LOCKED
Deactivation	Subsystem LOCKED	Subsystem LOCKED
Subsystem session (work task)	INIT routine called	Subsystem LOCKED
• when swapping subsystem versions and interrupting availability		
Activation of V2	Subsystem V2 LOCKED and swap aborted	Subsystem V2 LOCKED and swap aborted
STOPCOM routine of V1	Subsystem V1 LOCKED and INIT routine for V2 called	Subsystem V1 LOCKED and INIT routine for V2 called
INIT routine of V2	INIT routine of V2 called and DEINIT routine of V1 continued	Subsystem V2 LOCKED and DEINIT routine of V1 continued
Deactivation of V1	Subsystem V1 LOCKED	Subsystem V1 LOCKED
DEINIT routine of V1	Subsystem V1 LOCKED	Subsystem V1 LOCKED
• when swapping subsystem versions without interrupting availability		
Activation of V2	Subsystem V2 LOCKED and swap aborted	Subsystem V2 LOCKED and swap aborted
CLOSE-CTRL routine of V1	Swap aborted (subsystem V1 CREATED)	Subsystem V1 LOCKED and unloading of V1

INIT routine of V2	INIT routine of V2 called and DEINIT routine of V1 continued	Subsystem V2 LOCKED and swap aborted
STOPCOM routine of V1	Subsystem V1 LOCKED	Subsystem V1 LOCKED
DEINIT routine of V1	Subsystem V1 LOCKED	Subsystem V1 LOCKED
Deactivation of V1	Subsystem V1 LOCKED	Subsystem V1 LOCKED

Error log SERSLOG

DSSM writes an entry in the SERSLOG error log whenever

- a system call is faulty,
- inconsistent data is detected in the internal subsystem catalog or
- the DSSM task terminates abnormally (in this case, the SERSLOG entry contains a message describing the current situation).

The entry may consist of up to three parts:

1. the parameter list of the faulty system call or the inconsistent data (in 2K units),
2. the return code (if it is not already included in the parameter list) and
3. the address of the faulty routine and the address of the routine that called the faulty routine. This entry is useful for diagnostic purposes because DSSM uses its own central SERSLOG call.

3.8 DSSM commands

Command	Meaning
ADD-SUBSYSTEM	Extend dynamic subsystem catalog
HOLD-SUBSYSTEM	Place subsystem in wait state
MODIFY-SUBSYSTEM-PARAMETER	Modify subsystem parameters
RELEASE-SUBSYSTEM-SPACE	Release reserved address space for subsystems
REMOVE-SUBSYSTEM	Remove inactive subsystem from dynamic catalog
RESUME-SUBSYSTEM	Cancel wait state for subsystem
SAVE-SUBSYSTEM-CATALOG	Save changes to dynamic subsystem catalog
SET-DSSM-OPTIONS	Activate/deactivate DSSM logging function
SHOW-DSSM-INFORMATION	Show information on DSSM
SHOW-RESTART-OPTIONS	Display information on automatic restart
SHOW-SUBSYSTEM-ATTRIBUTES	Request information on subsystem attributes
SHOW-SUBSYSTEM-INFO	Request information on current subsystem configuration
SHOW-SUBSYSTEM-STATUS	Request information on status of subsystems
START-SUBSYSTEM	Activate subsystem
STOP-SUBSYSTEM	Deactivate subsystem
UNLOCK-SUBSYSTEM	Shift subsystem from LOCKED status to NOT-CREATED status

Table 7: DSSM commands

The commands are described in the manual „BS2000 OSD/BC“. Here you also find the **SDF syntax representation** of the commands.

3.9 Example for output in an S variable

This example illustrates output in S variables as a function of the privileges of the caller. It does not represent a situation lifted from actual practice; it is rather an artificial, simplified example designed specially to show the principles involved.

The following privileges exist:

- Case 1: The caller has only the privilege STD-PROCESSING
- Case 2: The caller has the privileges STD-PROCESSING and SUBSYSTEM-MANAGEMENT or OPERATING
- Case 3: The caller does not have the privilege STD-PROCESSING, but does have at least one of the privileges SUBSYSTEM-MANAGEMENT and OPERATING

The following configuration exists:

- the global nonprivileged subsystem AA V04.6 has the status CREATED; one task is connected (TID=00010054, TSN=0123)
- the global, privileged subsystem XX V01.0 has the status CREATED: one task is connected (TID=0002006F, TSN=0BFC)
- the global, privileged subsystem XX V02.0 has the status IN-DELETE: two tasks are connected (TID=00070015, TSN=0CMM and TID=00010057, TSN=00AP)

A compound variable of the type “list” with the name DATA is declared and assigned to the S variable stream SYSINF.

```
/DECLARE-VARIABLE VAR-NAME=DATA(TYPE=*STRUCTURE),
MULTIPLE-ELEMENTS=*LIST

/ASSIGN-STREAM STREAM-NAME=SYSINF,TO=*VAR(VAR-NAME=DATA)

/SHOW-STREAM-ASSIGNMENT

       STREAM-NAME   = SYSINF
        ASSIGN-LEVEL = 0

        DESTINATION  = *VARIABLE

          VARIABLE-NAME = DATA
               VAR-MODE = *EXTEND

          RETURN-VARIABLE-NAME = *NONE

          CONTROL-VAR-NAME = *NONE
          RET-CONTROL-VAR-NAME = *NONE

    STREAM-NAME   = SYSMSG
```

Case 1: The caller has only the privilege **STD-PROCESSING**

```
/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=*ALL ──────────────────────  (1)

% LOCAL SUBSYSTEM AA     /V04.5    IS CREATED

% SUBSYSTEM AA     /V04.6    IS CREATED
/SHOW-VAR DATA ──────────────────────────────────────────────────────  (2)

   :

   DATA(*LIST).SUBSYS-TYPE='*LOC'
   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.5'

   DATA(*LIST).SUBSYS-STA='*CREATED'
   DATA(*LIST).SUBSYS-TYPE='*GLB'

   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.6'
   DATA(*LIST).SUBSYS-STA='*CREATED'

/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=AA ────────────────────────  (3)

% LOCAL SUBSYSTEM AA     /V04.5    IS CREATED

% SUBSYSTEM AA      /V04.6    IS CREATED

/SHOW-VAR DATA ──────────────────────────────────────────────────────  (4)
   :

   DATA(*LIST).SUBSYS-TYPE='*LOC'

   DATA(*LIST).SUBSYS-NAME='AA'
   DATA(*LIST).SUBSYS-VERSION='04.5'

   DATA(*LIST).SUBSYS-STA='*CREATED'

   DATA(*LIST).SUBSYS-TYPE='*GLB'
   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.6'

   DATA(*LIST).SUBSYS-STA='*CREATED'
```

(1) The names, versions and statuses of all global, nonprivileged subsystems and of all local subsystems are output to SYSOUT. In our specially constructed example, only the global subsystem AA is nonprivileged. It is output - together with the local subsystem AA.

(2) Output in the S variable DATA explicitly contains the subsystem type “global” or “local”.

(3) The restriction concerning output to subsystem AA does not result in any changes in the output to SYSOUT, in contrast to (1).

(4) The restriction concerning output to subsystem AA does not result in any changes in the output to the S variable DATA in contrast to (2).

Case 2: The caller has the privileges **STD-PROCESSING** and **SUBSYSTEM-MANAGEMENT** and/or **OPERATING**

```
/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=*ALL ──────────────────────────  (5)

% LOCAL SUBSYSTEM AA     /V04.5    IS CREATED

% SUBSYSTEM AA     /V04.6    IS CREATED
% SUBSYSTEM XX     /V01.0    IS CREATED

% SUBSYSTEM XX     /V02.0    IS IN DELETE / WAIT-DISCON

/SHOW-VAR DATA ──────────────────────────────────────────────────────  (6)
   :

   DATA(*LIST).SUBSYS-TYPE='*LOC'

   DATA(*LIST).SUBSYS-NAME='AA'
   DATA(*LIST).SUBSYS-VERSION='04.5'

   DATA(*LIST).SUBSYS-STA='*CREATED'

   DATA(*LIST).SUBSYS-TYPE='*GLB'
   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.6'

   DATA(*LIST).SUBSYS-STA='*CREATED'
   DATA(*LIST).SUBSYS-TYPE='*GLB'

   DATA(*LIST).SUBSYS-NAME='XX'

   DATA(*LIST).SUBSYS-VERSION='01.0'
   DATA(*LIST).SUBSYS-STA='*CREATED'

   DATA(*LIST).SUBSYS-TYPE='*GLB'

   DATA(*LIST).SUBSYS-NAME='XX'

   DATA(*LIST).SUBSYS-VERSION='02.0'
   DATA(*LIST).SUBSYS-STA='*IN-DELETE'

   DATA(*LIST).SUBSYS-INT-STA='WAIT-DISCON'

/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=AA ────────────────────────────  (7)
% LOCAL SUBSYSTEM AA     /V04.5    IS CREATED

% SUBSYSTEM AA     /V04.6    IS USED BY 1 TASK

% TASKID 00010054

% TSN        0123

%   4 CONNECTIONS SINCE STARTUP
```

```
/SHOW-VAR DATA ──────────────────────────────────────────────────────────── (8)
   :

   DATA(*LIST).SUBSYS-TYPE='*LOC'

   DATA(*LIST).SUBSYS-NAME='AA'
   DATA(*LIST).SUBSYS-VERSION='04.5'

   DATA(*LIST).SUBSYS-STA='*CREATED'

   DATA(*LIST).SUBSYS-TYPE='*GLB'
   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.6'

   DATA(*LIST).SUBSYS-STA='*CREATED'
   DATA(*LIST).CONN-NUM-SINCE-START=4

   DATA(*LIST).USED-TASK-LIST(*LIST).TID='00010054'

   DATA(*LIST).USED-TASK-LIST(*LIST).TSN='0123'
```

(5) The names, versions and statuses of all global and local subsystems are output to SYSOUT. In other words, information is provided on all subsystems loaded in the current system. In our specially constructed example, this refers to the global subsystems XX (privileged, two versions) and AA (nonprivileged) and to the local subsystem AA.

(6) Output in the S variable DATA explicitly contains the subsystem type “global” or “local”.

(7) The restriction affecting the output to subsystem AA does not result in any changes as compared with (1): any tasks connected to a subsystem are shown together with their TSN and TID. The total number of tasks connected to this subsystem since startup is likewise output to SYSOUT.

(8) Output from the subsystems AA in the S variable DATA contains the same information as in the output to SYSOUT, as well as additional information concerning the subsystem type “global” or “local”.

Case 3: The caller does not have the privilege STD-PROCESSING but does have one of the privileges **SUBSYSTEM-MANAGEMENT** or **OPERATING**

```
/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=*ALL ──────────────────────────  (9)
% SUBSYSTEM AA     /V04.6    IS CREATED

% SUBSYSTEM XX     /V01.0    IS CREATED

% SUBSYSTEM XX     /V02.0    IS IN DELETE / WAIT-CLS-CTRL
/SHOW-VAR DATA ──────────────────────────────────────────────────────── (10)

   :

   DATA(*LIST).SUBSYS-TYPE='*GLB'
   DATA(*LIST).SUBSYS-NAME='AA'

   DATA(*LIST).SUBSYS-VERSION='04.6'

   DATA(*LIST).SUBSYS-STA='*CREATED'
   DATA(*LIST).SUBSYS-TYPE='*GLB'

   DATA(*LIST).SUBSYS-NAME='XX'

   DATA(*LIST).SUBSYS-VERSION='01.0'
   DATA(*LIST).SUBSYS-STA='*CREATED'

   DATA(*LIST).SUBSYS-TYPE='*GLB'

   DATA(*LIST).SUBSYS-NAME='XX'
   DATA(*LIST).SUBSYS-VERSION='02.0'

   DATA(*LIST).SUBSYS-STA='*IN-DELETE'

   DATA(*LIST).SUBSYS-INT-STA='WAIT-CLS-CTRL'

/SHOW-SUBSYSTEM-STATUS SUBSYSTEM-NAME=XX ──────────────────────────── (11)

% SUBSYSTEM XX     /V0.0     IS USED BY 1 TASK

% TASKID 0002006F

% TSN        0BFC

%   7 CONNECTIONS SINCE STARTUP

% SUBSYSTEM XX     /V02.0    IS USED BY 2 TASKS
% TASKID 00070015 00010057

% TSN        0CMM     0OAP

%                        IS IN DELETE / WAIT-DISCON
% 12 CONNECTIONS SINCE STARTUP

/SHOW-VAR DATA ──────────────────────────────────────────────────────── (12)

   :
   DATA(*LIST).SUBSYS-TYPE='*GLB'
```

```
DATA(*LIST).SUBSYS-NAME='XX'

DATA(*LIST).SUBSYS-VERSION='01.0'
DATA(*LIST).SUBSYS-STA='*CREATED'

DATA(*LIST).CONN-NUM-SINCE-START=7

DATA(*LIST).USED-TASK-LIST(1).TID='0002006F'
DATA(*LIST).USED-TASK-LIST(1).TSN='0BFC'

DATA(*LIST).SUBSYS-TYPE='*GLB'

DATA(*LIST).SUBSYS-NAME='XX'

DATA(*LIST).SUBSYS-VERSION='02.0'
DATA(*LIST).SUBSYS-STA='*IN-DELETE'

DATA(*LIST).SUBSYS-INT-STA='WAIT-DISCON'

DATA(*LIST).CONN-NUM-SINCE-START=12
DATA(*LIST).USED-TASK-LIST(*LIST).TID='00070015'

DATA(*LIST).USED-TASK-LIST(*LIST).TSN='0CMM'

DATA(*LIST).USED-TASK-LIST(*LIST).TID='00010057'
DATA(*LIST).USED-TASK-LIST(*LIST).TSN='00AP'
```

(9) The names, versions and statuses of all global subsystems are output to SYSOUT. In our specially constructed example, this refers to the global subsystems XX (privileged, two versions) and AA (nonprivileged).

(10) Output in the S variable DATA contains the same information as in output to SYSOUT.

(11) The restriction concerning output to the subsystem XX produces the following changes as compared with (9): any tasks connected to a subsystem are displayed with their TSN and TID. The total number of tasks connected to this subsystem since startup is likewise output to SYSOUT.

(12) Output from the subsystems XX in the S variable DATA contains the same information as in output to SYSOUT, with the addition of information concerning the subsystem type “global”.

4 SSCM

The subsystem SSCM (Static Subsystem Catalog Manager) ensures flexible, user-friendly management of the static subsystem catalogs (SSMCAT).

For the procedure for installing SSCM and for information on the coexistence of SSCM versions, see section “Installing SSCM”.

4.1 Generating a subsystem catalog

The subsystem catalog which is to be created must be placed on the home pubset and stored under the TSOS user ID. The catalog may be given any name desired, and the name can be declared to the system by means of the parameter service (see "Startup of dynamic subsystem management"). The program provided for generating a subsystem catalog is SSCM.
The diagram on the following page shows the procedure for generating a subsystem catalog.

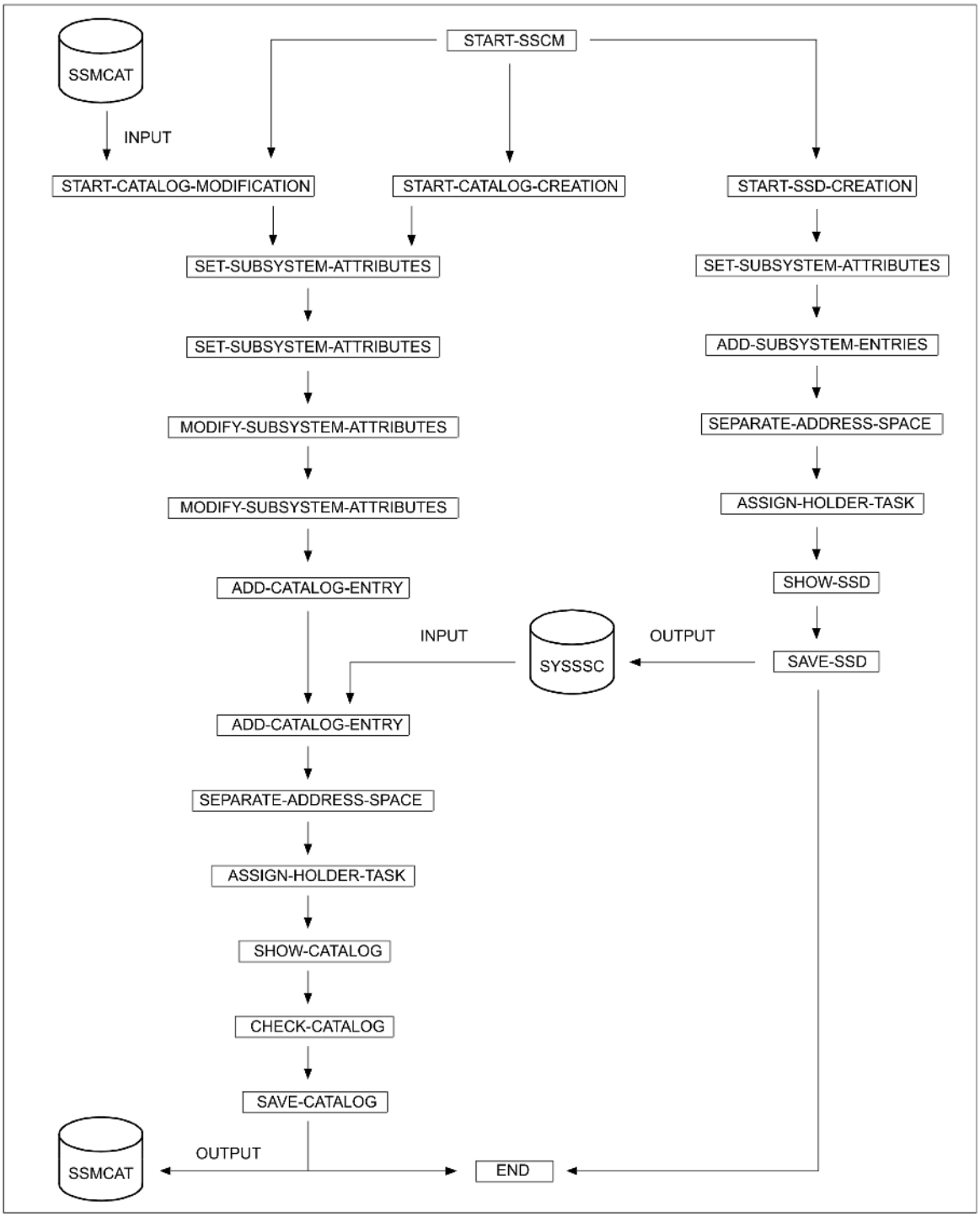


Figure 1: Generating a subsystem catalog

4.2 Starting and terminating SSCM

To start the SSCM program use following command.

START-**SSCM**
MONJV = ***NONE** / <filename 1..54 without-gen-vers> ,**CPU-**LIMIT = ***JOB-REST** / <integer 1..32767>

You can also call up SSCM using the abbreviation **SSCM**.

The message file for SSCM ($TSOS.SYSMES.SSCM.210) is activated and the SSCM link load module in the SYSLNK.SSCM.210 module library is loaded (message `BLS0517`).

To terminate SSCM, use the **END** statement.

Monitoring programs by means of monitoring job variables

A job variable which is to be used to monitor a program must be declared in the START-SSCM command with MONJV=<jv-name>.

SSCM can set the monitoring variable to the following values:

`'$T0000'`	The program terminated normally.
`'$T1010'`	Statement rejected; execution of the program will be continued.
`'$A2010'`	Statement rejected; the program has been terminated.
`'$A2015'`	Unexpected EOF on SYSDTA; the program has been terminated.
`'$A3020'`	SSCM-internal error; the program is being terminated.

For details of program monitoring by means of monitoring job variables refer also to the “BS2000 OSD/BC" and “Job Variables” manuals.

Monitoring by means of task switches

If a statement is abnormally terminated by SSCM, task switch 31 is activated.

If task switch 1 is activated, SSCM does not execute instructions and will merely execute only a syntax check.

4.3 The SSCM statements

SSCM is offered with an SDF interface for the user interface. This gives the SSCM user all the functions and advantages offered by SDF - guided dialog, help texts for operands, the use of default values.

The **SDF syntax representation** of the commands is explained in the manual „B2000 OSD/BC Commands“.

The following SSCM statements are available to (sub)system administration:

Command	Meaning
ADD-CATALOG-ENTRY	Add subsystem definition(s) to subsystem catalog
ADD-SUBSYSTEM-ENTRIES	Define additional job entry points
ASSIGN-HOLDER-TASK	Distribute subsystems to holder tasks
CHECK-CATALOG	Check subsystem definition(s) for consistency
GENERATE-CATALOG-SOURCE	Create SSCM statement list for generation
MODIFY-SUBSYSTEM-ATTRIBUTES	Modify subsystem attributes
MODIFY-WORK-TASK-ATTRIBUTE	Modify work task parameters
REMOVE-ADDR-SPACE-SEPARATION	Revoke disjunctive distribution of subsystems in class 5 memory
REMOVE-CATALOG-ENTRY	Logically delete definition of subsystem from subsystem catalog
SAVE-CATALOG	Save subsystem catalog as PAM file
SAVE-SSD	Terminate subsystem definition(s)
SEPARATE-ADDRESS-SPACE	Control disjunctive distribution of subsystems in class 5 memory
SET-SUBSYSTEM-ATTRIBUTES	Define attributes and entry points of subsystem
SHOW-CATALOG	Show subsystem configuration
SHOW-SSD	Show contents of SSD object (subsystem definitions)
START-CATALOG-CREATION	Define name of static subsystem catalog
START-CATALOG-MODIFICATION	Modify static subsystem catalog
START-SSD-CREATION	Generate SSD object for adding subsystem definitions

Table 8: SSCM statements

4.3.1 ADD-CATALOG-ENTRY

Add subsystem definition(s) to subsystem catalog

Function

This statement copies the definition of new subsystems, which are held in an SSD object, into the subsystem catalog which is currently open.

This statement will be rejected if the catalog into which the definitions are to be integrated is not currently open as a result of a START-CATALOG-CREATION or START-CATALOG-MODIFICATION statement. Any definition for a subsystem which is already contained in the open catalog will be ignored; processing will continue, however, after output of a corresponding message.

Format

ADD-CATALOG**-ENTRY**
FROM-FILE = <filename 1..54 without-gen-vers>
,**INSTAL**LATION**-USERID** = ***UNCHA** NGED / ***DEF**AULT**-USERID** / <name 1..8>
,**CORR**ECTION**-STATE** = ***UNCHA** NGED / <c-string 3..3> / <text 3..3>

Operands

FROM-FILE = <filename 1..54>
Name of the file in which to search for the subsystem definition(s). This file must be an SSD object generated by SSCM of type ISAM, in which the attributes of one or more subsystems are stored.

INSTALLATION-USERID =
Specifies a user ID under which the subsystem satellites (REP file, object module library, message file, syntax file and subsystem information file) are expected, in cases where these files have not already been assigned to a user ID.

INSTALLATION-USERID = *UNCHANGED
Default value: the files are expected under the user ID specified in the subsystem definition (SET-SUBSYSTEM-ATTRIBUTES statement).

INSTALLATION-USERID = *DEFAULT-USERID
The files are expected under the system's default user ID (prefix "$.").

INSTALLATION-USERID = <name 1..8>
User ID under which the files are expected. If a different ID was specified in the SET-SUBSYSTEM-ATTRIBUTES statement for an SSD object, it will be overwritten.

CORRECTION-STATE =
Replaces in the catalog the last three characters of the subsystem version from the SSD file; these characters indicate the release and correction states of the subsystem version. If the subsystem version in the SSD file consists of four characters, the three characters specified for CORRECTION-STATE are chained with them.

CORRECTION-STATE = *UNCHANGED
Default value: the release and correction states remain unchanged.

CORRECTION-STATE = <text 3..3>
Specifies the release and correction states in the format: ann, in which the text elements have the following meanings:

a: Release state; alphabetic character
nn: Correction state; numeric characters

CORRECTION-STATE = <c-string 3..3>
Specifies the release and correction states as a character string in the format: ann;for the meanings of the text elements, see CORRECTION-STATE=<text 3..3>.

Notes

- If a subsystem definition contains file names without a user ID and if no installation ID is specified, the relevant DSSM task searches for the files under the user ID TSOS or, in the case of a local subsystem, under the user ID of the calling task.
  If the files are stored under a different user ID, this ID must be specified in the INSTALLATION-USERID operand of one of the commands ADD-CATALOG-ENTRY, SET-SUBSYSTEM-ATTRIBUTES or MODIFY-SUBSYSTEM-ATTRIBUTES.
- If an SSD object containing multiple subsystem definitions is specified in the FROM-FILE operand, the release and correction states of all the subsystems defined in this SSD object will be affected by the value of the CORRECTION-STATE and INSTALLATION-USERID operands.
- The file names of a subsystem are not changed if the release and correction states are specified. Users wishing to change these names must use the MODIFY-SUBSYSTEM-ATTRIBUTES statement to do so.
- If the CORRECTION-STATE operand is specified, a version check for mutually dependent subsystems or for subsystems which have address relations with one another may result in a CHECK-CATALOG error. This can be avoided by specifying suitable entries in
  MODIFY-SUBSYSTEM-ATTRIBUTES MODIFY-REFER-SUBS=..., MODIFY-RELATED-SUBS=....

4.3.2 ADD-SUBSYSTEM-ENTRIES

Define additional job entry points

Function

This statement can be used to define further job entry points for a subsystem in an object module file (SSD object) in addition to those already specified in SET-SUBSYSTEM-ATTRIBUTES, even exceeding the maximum permitted number of 100.

Each ADD-SUBSYSTEM-ENTRIES statement (which can be executed repeatedly for one and the same subsystem) can be used to define up to 100 additional job entry points for a single subsystem.

ADD-SUBSYSTEM-ENTRIES is rejected if no START-SSD-CREATION statement was executed beforehand.

The statement is rejected if the specified subsystem was defined with the SET-SUBSYSTEM-ATTRIBUTES or the MODIFY-SUBSYSTEM-ATTRIBUTES statement with entry points that are to be supplied dynamically (*BY-PROGRAM(...)).

Execution of the statement is aborted

- if the subsystem specified by TO-SUBSYSTEM and VERSION is not found in the current SSD object or
- if an existing job entry point is to be defined a second time.

An error message is issued.

Format

ADD-SUBSYSTEM-ENTRIES

TO-SUBSYSTEM = <structured-name 1..8>(...)

<structured-name 1..8>(...)

VERSION = <c-string 3..8> / <text 3..8> “

,**SUBSYSTEM-ENTRIES** = list-poss(100): <text 1..8>(...)

<text 1..8>(...)

MODE = ***LINK** / ***ISL**(...) / ***SVC**(...) / ***SYSTEM-EXIT**(...)

***ISL**(...)

FUNCTION-NUMBER = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SVC**(...)

NUMBER = <integer 0..255>

,**CALL-BY-SYSTEM-EXIT** = ***ALLOW** ED / ***FORBIDDEN**

,**FUNCTION-NUMBER** = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SYSTEM-EXIT**(...)

NUMBER = <integer 0..127>

,**CONNECTION-ACCESS** = ***ALL** / ***SYS**TEM / ***SIH**

,**CONNECTION-SCOPE** = ***TASK** / ***PROG**RAM / ***FREE** / ***CALL** / ***OPTIMAL**

, **FIRST-CONNECTION** = ***ALLOW** ED / ***FORBIDDEN**

Operands

TO-SUBSYSTEM = <structured-name 1..8>(...)
Specifies the name and version of the subsystem for which additional job entry points are to be defined.

VERSION = <c-string 3..8> / <text 3..8>
Specifies the version of the subsystem in the format “[V][n]n.m[ann]”. The text elements have the following meanings:

nn = Main version (numeric)
m = Revision version (numeric)
ann = Update status (a=letter, release status; nn=numeric, correction status)

SUBSYSTEM-ENTRIES =
Declares additional entry points (job entry points) which are to be associated with the subsystem. Up to 100 job entry points can be defined per statement.

SUBSYSTEM-ENTRIES = list-poss(100): <text 1..8>
Specifies up to 100 job entry points by name; the type of each entry point must be defined in the substructures.

MODE =
Defines the type of a job entry which is defined for the subsystem.

MODE = *LINK
Default value: the job entry cannot be accessed by indirect linkage, but only by using a CONNECT relation through an external linkage editor symbol.
In the case of different versions of the same subsystem which use the same external linkage editor symbol, DSSM automatically sets up a link to the highest loaded version of the subsystem.

MODE = *ISL(...)
The job entry is implemented by indirect linkage via System Procedure Linkage (for privileged subsystems only). If the specification includes in addition a function and version number for the ISL entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem (if VERSION-COEXISTENCE=*ALLOWED is specified, see the SET-SUBSYSTEM-ATTRIBUTES statement).
For different subsystems, if the job entry is to be accessed by the same ISL entry point, they must be uniquely identified by specifying the function and version numbers.
In the case of different versions of the same subsystem which use the same ISL entry point, then - if the function and version numbers are not specified - DSSM will automatically set up a connection to the highest loaded version of the subsystem.
In the case of different versions of the same subsystem which use the same ISL entry point and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller’s parameter list.
The specification CONNECTION-ACCESS=*ALL (see the SET-SUBSYSTEM-ATTRIBUTES statement) is not permissible for ISL entry points.

FUNCTION-NUMBER =
Specifies whether a particular function and version number of the ISL entry point is to be addressed, because the same ISL entry point can be used by different functions.

FUNCTION-NUMBER = *NONE
Default value: no particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)
Number of the ISL entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>
Version of the specified ISL function number.

MODE = *SVC(...)

Job entry is to be effected by an indirect connection using a supervisor call (SVC).
If the specification includes in addition a function and version number for the SVC entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem
(if VERSION-COEXISTENCE=*ALLOWED is specified, see the SET-SUBSYSTEM-ATTRIBUTES statement).
For different subsystems, if the job entry is to be accessed by the same SVC, they must be uniquely identified by specifying the function and version numbers.
In the case of different versions of the same subsystem which use the same SVC, then - if the function and version numbers are not specified - DSSM will automatically set up a connection to the highest loaded version of the subsystem.
In the case of different versions of the same subsystem which use the same SVC and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller's parameter list.
If this operand value is specified, it is better to set the operand CONNECTION-ACCESS to the value *SYSTEM, instead of *ALL (see the SET-SUBSYSTEM-ATTRIBUTES statement).

NUMBER = <integer 0..255>

Number of the SVC via which job entry is to be effected. No SVC number greater than 191 may be used in conjunction with CONNECTION-ACCESS=*ALL (see the SET-SUBSYSTEM-ATTRIBUTES statement).

CALL-BY-SYSTEM-EXIT =

Defines whether the specified SVC number may be called from within system exit routines.

CALL-BY-SYSTEM-EXIT = *ALLOWED

Default value: system exit routines are permitted to call the specified SVC number.

CALL-BY-SYSTEM-EXIT = *FORBIDDEN

System exit routines are not permitted to call the specified SVC number.

FUNCTION-NUMBER =

Specifies whether a particular function and version number of the SVC entry point is to be addressed, because the same SVC entry point can be used by different functions.

FUNCTION-NUMBER = *NONE

Default value: no particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)

The number of an SVC entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>

Version of the specified SVC function number.

MODE = SYSTEM-EXIT(...)

Job entry is to be effected by an indirect connection using system exit routines.
This operand must not be used in conjunction with CONNECTION-ACCESS=*ALL
(see the SET-SUBSYSTEM-ATTRIBUTES statement).

NUMBER = <integer 0..127>

Number of the system exit routine.

CONNECTION-ACCESS =

Specifies the access authorization (privileges) required by the subsystem.

CONNECTION-ACCESS = *ALL
Default value: privileged and nonprivileged program runs may access the subsystem.This operand value must not be used in conjunction with MODE=*SYSTEM-EXIT/*ISL/*SVC (with an SVC number greater than 191; see the SET-SUBSYSTEM-ATTRIBUTES statement).

CONNECTION-ACCESS = *SYSTEM
Only privileged program runs may access the subsystem.

CONNECTION-ACCESS = *SIH
Only tasks running in the SIH processor state may access the subsystem.

The subsystem called also runs in the SIH processor state, i.e. it is uninterruptible.This operand value is permissible only for subsystems for which the entry point is defined via:

- System Procedure Linkage (MODE=*ISL(FUNCTION-NUMBER=*NONE))
- CONNECTION-SCOPE=*OPTIMAL
- MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*SYSTEM)

CONNECTION-SCOPE =
Identifies the event which will call up the automatic cleardown of the connection to the specified subsystem job entry point.

CONNECTION-SCOPE = *TASK
Default value: the connection will be cleared when the task terminates.

CONNECTION-SCOPE = *PROGRAM
The connection will be cleared when the program terminates, or before.
Only CONNECTION-SCOPE=*PROGRAM may be specified in conjunction with MEMORY-CLASS=*LOCAL-UNPRIVILEGED (see the SET-SUBSYSTEM-ATTRIBUTES statement).
This operand value is recommended for subsystems which were declared with SUBSYSTEM-ACCESS=*LOW/*HIGH.

CONNECTION-SCOPE = *FREE
DSSM is not to carry out any checking of the connections to the job entry point. The connection will not be automatically cleared - unless explicitly requested. To avoid problems or possible errors when the subsystem is being unloaded, the connections must be managed by the subsystem itself.

CONNECTION-SCOPE = *CALL
On return from this job entry point, DSSM will automatically clear the connections.
This operand value is only available with subsystems for which the job entry is defined by means of System Procedure Linkage (ISL) or supervisor calls (SVC).

CONNECTION-SCOPE = *OPTIMAL
The subsystem is deactivated or suspended when there are no further tasks with a connection to this job entry point.
A routine with an entry point defined with *OPTIMAL must be terminated with RETURN.If an entry point of a subsystem is defined with CONNECTION-SCOPE=*OPTIMAL, all of its entry points should be defined in the subsystem catalog with MODE!=*LINK. While a subsystem is deactivated or suspended, no call of the subsystem with CONNECTION-SCOPE=*OPTIMAL is accepted.
Exception: if the subsystem was defined with CREATION-TIME=*AT-SUBSYSTEM-CALL and the calling task is already connected to the subsystem (see the SET-SUBSYSTEM-ATTRIBUTES statement).

FIRST-CONNECTION =
Determines whether or not first connection of the task to the specified job entry point in the subsystem is allowed. At least one job entry point of a subsystem must be defined with FIRST-CONNECTION=*ALLOWED.

FIRST-CONNECTION = *ALLOWED
Default setting: first connection to the specified job entry point is allowed.

FIRST-CONNECTION = *FORBIDDEN
Connection to the specified job entry point via SVC or ISL is not allowed if the task has not yet been connected to another job entry point belonging to the subsystem.
It is not permitted to specify this operand value for job entry points that have been defined with MODE=*LINK /*SYSTEM-EXIT or CONNECTION-ACCESS=*SIH.

4.3.3 ASSIGN-HOLDER-TASK

Distribute subsystems to holder tasks

Function

This statement is used to control the distribution of subsystems to holder tasks. The subsystems listed in the statement can use the holder task as a work task, or will be set up together in a holder task. If this statement is omitted when creating a catalog, SSCM will make a standard assignment of the subsystems in order to limit the number of holder tasks.

This statement is mandatory for subsystems which use the holder task as a work task.

In order to become a work task, the entry points of the subsystem must be defined with one of the following combinations:

CLOSE-CTRL	STOPCOM	DEINIT
*DYNAMIC	*DYNAMIC	*DYNAMIC
*DYNAMIC	*NO	*DYNAMIC
*NO	*DYNAMIC	*DYNAMIC *
*NO	*NO	*DYNAMIC *

*For compatibility reasons it is not obligatory to define a DEINIT routine for a subsystem without a CLOSE-CTRL routine. It should be noted, however, that no guarantee can be given for correct execution of the subsystem in such a case.

The statement may only be used once for an SSD object. If there are two consecutive statements for the same subsystem and there is a conflict (i.e. contradictory declarations), the definition which requires the holder task to be used as a work task will take precedence.A declaration referring to a shared holder task applies to every version of the subsystem, except for any versions which use the holder task as a work task.

ASSIGN-HOLDER-TASK is rejected if none of the following statements has been executed beforehand:

- START-SSD-CREATION
- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

Format

ASSIGN-HOLDER-TASK

TYPE = **<u>*WORK-TASK</u>** (...) / ***SHARED-HOLDER**(...)

***WORK-TASK**(...)

SUBSYSTEM-NAME = <structured-name 1..8>

,**SUBSYS**TEM**-VERS**ION = <c-string 3..8> / <text 3..8>

,**TSN** = **<u>*BY-DSSM</u>** / <alphanum-name 1..4>

***SHARED-HOLDER**(...)

BY-SUBSYSTEMS = list-poss(15): <structured-name 1..8>

,**TSN** = **<u>*BY-DSSM</u>** / <alphanum-name 1..4>

Operands

TYPE =
Specifies whether the holder task is to be used as a work task or the subsystem is to be created in a shared holder task.

TYPE = <u>*WORK-TASK</u>(...)
Default value: the holder task is to be used as a work task.

SUBSYSTEM-NAME = <structured-name 1..8>
Name of the subsystem which is to use the holder task as a work task.

SUBSYSTEM-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to use the holder task as a work task.
This version must have been declared previously.

TSN =
Specifies the task sequence number (TSN) to be given to the subsystem’s work task.

TSN = <u>*BY-DSSM</u>
Default value: the TSN will be issued by DSSM when the work task is loaded. The TSN is known, because is calculated by DSSM. The SYSLST output of the SHOW-SUBSYSTEM-INFO command and the list variable output of the SHOW-SUBSYSTEM-ATTRIBUTES command will both display the TSN value, regardless of the subsystem's creation status.

TSN = <alphanum-name 1..4>
The TSN to be given to the work task when it is started.
The specified TSN must be uniquely defined and usable when the subsystem is loaded.

TYPE = *SHARED-HOLDER(...)
The subsystem is to be created in a shared holder task.

BY-SUBSYSTEMS = list-poss(15): <structured-name 1..8>
Names of up to 15 subsystems which are to be created in the same holder task.The first of the subsystems in the list must have been declared previously.

TSN =
Specifies the task sequence number (TSN) to be given to the shared holder task.

TSN = *BY-DSSM

Default value: the TSN will be issued by DSSM when the work task is loaded. The TSN is known, because is calculated by DSSM. If at least one of the subsystems in this shared holder task is in CREATED status - TSN is already issued. The SYSLST output of the SHOW-SUBSYSTEM-INFO command and the list variable output of the SHOW-SUBSYSTEM-ATTRIBUTES command will both display the TSN value, regardless of the subsystem's creation status.

TSN = <alphanum-name 1..4>

Task sequence number to be given to the shared holder task when it is started.
The specified TSN must be uniquely defined and usable when the subsystem is loaded.

4.3.4 CHECK-CATALOG

Check subsystem definition(s) for consistency

Function

This statement is used to carry out consistency checks on the definitions of subsystems held in a catalog.

CHECK-CATALOG will be rejected if the file name specified by the user does not exist, or if the subsystem catalog is empty.

If the specified file name does not correspond with that of the catalog which is currently open, the following message will be output:

```
SCM0011    DO YOU REALLY WANT TO OVERWRITE MEMORY CATALOG '(&00)'? REPLY (Y/N)
```

If the user replies with **Y**, the virtual definitions in the current catalog will be lost. If the reply is **N**, execution of the CHECK-CATALOG statement will be aborted, and the user can then use the SAVE-CATALOG statement to save to a file all subsystem definitions which have not yet been saved.

> The catalog cannot be saved without first carrying out checks on any link and dependency relations.

Format

CHECK-CATALOG
CATALOG-NAME = ***CURR** ENT / <filename 1..54 without-gen-vers>
,**DEPENDENCE-RELATION** = ***Y** ES / ***NO**
,**LINK-RELATION** = ***Y** ES / ***NO**
,**RELATED-FILES** = ***NO** / ***Y**ES
,**OUTPUT** = ***SYSOUT** / ***SYSLST**(...)
***SYSLST**(...)
SYSLST-NUMBER = ***STD** / <integer 1..99>

Operands

CATALOG-NAME =
Specifies the name of the catalog which holds the definitions which are to be checked.

CATALOG-NAME = *CURRENT
Default value: the catalog which is currently open (START-CATALOG-CREATION or START-CATALOG-MODIFICATION statement) is to be checked.

CATALOG-NAME = <filename 1..54 without-gen-vers>
Fully qualified name of the static subsystem catalog whose contents are to be checked.

DEPENDENCE-RELATION = *YES / *NO
Specifies whether the checks on subsystem definitions are also to take into account dependency relations to other subsystems (*YES) or not (*NO). The catalog cannot be saved if *NO was specified in a prior CHECK-CATALOG statement.

LINK-RELATION = *YES / *NO
Specifies whether the checks on subsystem definitions are also to take into account address links to other subsystems (*YES, default value) or not (*NO). The catalog cannot be saved if *NO was specified in a prior CHECK-CATALOG statement.

RELATED-FILES = *NO / *YES
Specifies whether the existence of files which have dependency relations to these subsystems is to be checked (*YES) or not (*NO, default value).
If the names of dependent files were defined with the value *INSTALLED(...), the DEFAULT-NAME specified there will also be checked.

OUTPUT =
Specifies where to output the information generated by the statement, i.e. the results of the check run.

OUTPUT = *SYSOUT
Default value: the messages will be output to the terminal.

OUTPUT = *SYSLST(...)
The messages are to be output to SYSLST.

SYSLST-NUMBER =
Identifies the SYSLST file to which the output is to be directed.

SYSLST-NUMBER = *STD
Default value: output is to go to the default system file SYSLST.

SYSLST-NUMBER = <integer 1..99>
Output is to go to one of the system files from the set SYSLST01 to SYSLST99, the number of which must be specified here.

4.3.5 GENERATE-CATALOG-SOURCE

Create SSCM statement list for generation

Function

With this statement, SSCM creates a file containing a list of all SSCM statements required for (re)generation of a subsystem catalog (either for specific subsystems in the input catalog or for all of them).

Format

GENERATE-CATALOG-SOURCE
CATALOG-NAME = ***CURR**ENT / <filename 1..54 without-gen-vers> ,**SUBSYSTEM-NAME** = ***ALL** / <structured-name 1..8>(...) <structured-name 1..8>(...) **VERSION** = ***ALL** / <c-string 3..8> / <text 3..8> ,**OUTPUT** = ***SYSLST** (...) ***SYSLST**(...) **SYSLST-NUM**BER = ***STD** / <integer 1..99>

Operands

CATALOG-NAME =
Specifies the subsystem catalog in which the subsystem definitions are saved.

CATALOG-NAME = *CURRENT
Default setting. The current subsystem catalog is used.

CATALOG-NAME = <filename 1..54 without-gen-vers>
Name of the subsystem catalog.

SUBSYSTEM-NAME =
Subsystems whose definitions are to be output.

SUBSYSTEM-NAME = *ALL
Default setting. The definitions of all subsystems are to be output.

SUBSYSTEM-NAME = <structured-name 1..8>(...)
Name of the subsystem whose definition is to be output.

VERSION = *ALL / <c-string 3..8> / <text 3..8>
Version of the subsystem whose definition is to be output.

OUTPUT = *SYSLST(...)
System file to which the generated information is to be sent.

SYSLST-NUMBER = *STD
Default setting. The information will be sent to the SYSLST file.

SYSLST-NUMBER = <integer 1..99>

The information will be sent to the specified system file in the range SYSLST01 to SYSLST99.

4.3.6 MODIFY-SUBSYSTEM-ATTRIBUTES

Modify subsystem attributes

Function

This statement can be used to change any of the attributes and entry points which were defined in the SET-SUBSYSTEM-ATTRIBUTES statement.

When a definition is modified, the following points must be noted:

- the subsystem - identified by its name and version - must be present in the catalog which is currently open
- any attempt to add a job entry or relation which already exists will be rejected
- it is impermissible to attempt to modify or delete a job entry or relation which has not yet been defined
- the mode of a job entry may only be changed if all the parameters are specified; the default values *UNCHANGED will be rejected
- the memory class for a subsystem may only be changed if all the parameters are specified; the default values *UNCHANGED will be rejected

MODIFY-SUBSYSTEM-ATTRIBUTES is rejected if none of the following statements have been executed beforehand:

- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

Note on syntax

A special data type <symbol>, which is fully described in the “BLSSERV” manual, can also be used for the names of the entry points in the following operands (in the format, the data type is specified as <name>):

- LINK-ENTRY
- DYNAMIC-CHECK-ENTRY
- INIT-ROUTINE
- CLOSE-CTRL-ROUTINE
- STOPCOM-ROUTINE
- DEINIT-ROUTINE
- INTERFACE-VERSION
- ADD-SUBS-ENTRIES
- MODIFY-SUBS-ENTRIES
- REMOVE-SUBS-ENTRIES

Format

MODIFY-SUBSYSTEM-ATTRIBUTES
SUBSYSTEM-NAME = <structured-name 1..8>(...)   <structured-name 1..8>(...)   **VERSION** = <c-string 3..8> / <text 3..8>

,**INSTAL**LATION**-UNIT** = ***UNCHA** NGED / ***NONE** / ***STD** / <text 1..30>

,**INSTAL**LATION**-USERID** = ***UNCHA** NGED / ***NONE** / ***DEF**AULT**-USERID** / <name 1..8>

,**COPYRIGHT** = ***UNCHA** NGED / ***NONE** / <c-string 1..54>(...)

<c-string 1..54>(...)

YEAR = ***YEAR-1990** / <c-string 4..4>

,**LIB**RARY = ***UNCHA** NGED / ***STD** / ***CPLINK** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = ***** **UNCHA** NGED / <filename 1..30 without-catid-userid-gen-vers>

,**DEF**AULT**-NAME** = ***** **UNCHA** NGED / <filename 1..54 without-gen-vers>

,**SUBSYS**TEM**-LOAD-MODE** = ***UNCHA** NGED / ***STD** / ***ADVANCED**

,**REP-FILE** = ***UNCHA** NGED / ***STD** / ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = * **UNCHA** NGED / <filename 1..30 without-catid-userid-gen-vers>

,**DEF**AULT**-NAME** = ***** **UNCHA** NGED / <filename 1..54 without-gen-vers> / ***NONE**

,**REP-FILE-MANDATORY** = ***UNCHA** NGED / ***NO** / ***Y**ES

,**MES**SAGE**-FILE** = ***UNCHA** NGED / ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = ***** **UNCHA** NGED / <filename 1..30 without-catid-userid-gen-vers>

,**DEF**AULT**-NAME** = ***** **UNCHA** NGED / <filename 1..54 without-gen-vers> / ***NONE**

,**SUBSYS**TEM**-INFO-FILE** = ***UNCHA** NGED / ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = ***** **UNCHA** NGED / <filename 1..30 without-catid-userid-gen-vers>

,**DEF**AULT**-NAME** = ***** **UNCHA** NGED / <filename 1..54 without-gen-vers> / ***NONE**

,**SYNTAX-F**ILE = ***UNCHA** NGED / ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = ***** **UNCHA** NGED / <filename 1..30 without-catid-userid-gen-vers>

,**DEF**AULT**-NAME** = ***** **UNCHA** NGED / <filename 1..54 without-gen-vers> / ***NONE**

,**DYNAMIC-CHECK-ENTRY** = ***** **UNCHA** NGED / ***STD** / ***NO** / <text 1..8 without-sep>

,**CRE**ATION**-TIME** = ***UNCHA** NGED / ***AT-CRE**ATION**-REQ**UEST / ***AT-SUBSYS**TEM**-CALL**(...) /

***AT-DSSM-LOAD** / ***BEFORE-DSSM-LOAD** / ***MANDATORY-AT-STARTUP** /

***BEFORE-SYSTEM-READY** / ***AFTER-SYSTEM-READY**

***AT-SUBSYS**TEM**-CALL**(...)

ON-ACTION = ***STD** / ***ISL-CALL** / *****ANY

,**INIT-ROUTINE** = ***UNCHA** NGED / ***NO** / <text 1..8 without-sep>

,**CLOSE-CTRL-ROUTINE** = ***UNCHA** NGED / ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**STOPCOM-ROUTINE** = ***UNCHA** NGED / ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**DEINIT-ROUTINE** = ***UNCHA** NGED / ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**STOP-AT-SHUTDOWN** = ***UNCHA** NGED / ***NO** / ***Y**ES

,**INTERFACE-VERSION** = ***UNCHA** NGED / ***NO** / <text 1..8 without-sep>

,**SUBSYS**TEM**-HOLD** = ***UNCHA** NGED / ***ALLOW**ED / ***FORBID**DEN

,**STATE-CHANGE-CMDS** = ***** **UNCHA** NGED / ***ALLOW**ED / ***FORBID**DEN / ***BY-ADMINISTRATOR-ONLY**

,**FORCED-STATE-CHANGE** = ***UNCHA** NGED / ***ALLOW**ED / ***FORBID**DEN

,**RESET** = ***UNCHA** NGED / ***ALLOW**ED / ***FORBID**DEN

,**RESTART-REQUIRED** = ***UNCHA** NGED / ***NO** / ***Y**ES

,**VERSION-COEXISTENCE** = ***UNCHA** NGED / ***FORBID**DEN / ***ALLOW**ED

,**VERSION-EXCHANGE** = ***UNCHA** NGED / ***FORBID**DEN / ***ALLOW**ED

,**ADD-SUBS-ENTRIES** = ***NONE** / list-poss(100): <text 1..8>(...)

<text 1..8>(...)

MODE = ***LINK** / ***ISL**(...) / ***SVC**(...) / ***SYSTEM-EXIT**(...)

***ISL**(...)

FUNCTION-NUMBER = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SVC**(...)

NUMBER = <integer 0..255>

,**CALL-BY-SYSTEM-EXIT** = ***ALLOW** ED / ***FORBIDDEN**

,**FUNCTION-NUM**BER = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SYSTEM-EXIT**(...)

NUMBER = <integer 0..127>

,**CONNECTION-ACCESS** = ***ALL** / ***SYS**TEM / ***SIH**

,**CONNECTION-SCOPE** = ***TASK** / ***PROG**RAM / ***FREE** / ***CALL** / ***OPTIMAL**

,**FIRST-CONNECTION** = ***ALLOW** ED / ***FORBIDDEN**

,**MODIFY-SUBS-ENTRIES** = ***NONE** / list-poss(100): <text 1..8>(...) **/ *BY-PROGRAM(...)**

<text 1..8>(...)

MODE = ***UNCHA** NGED / ***LINK** / ***ISL**(...) / ***SVC**(...) / ***SYSTEM-EXIT**(...)

***ISL**(...)

FUNCTION-NUMBER = ***UNCHA** NGED (...) / ***NONE** / <integer 0..255>(...)

***UNCHA**NGED(...)

FUNCTION-VERSION = ***UNCHA** NGED / <integer 1..255>

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SVC**(...)

NUMBER = ***UNCHA** NGED / <integer 0..255>

,**CALL-BY-SYSTEM-EXIT** = ***UNCHA** NGED / ***ALLOW**ED / ***FORBIDDEN**

,**FUNCTION-NUMBER** = ***UNCHA** NGED (...) / ***NONE** / <integer 0..255>(...)

***UNCHA**NGED(...)

FUNCTION-VERSION = ***UNCHA** NGED / <integer 1..255>

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SYSTEM-EXIT**(...)

NUMBER = <integer 0..127>

,**CONNECTION-ACCESS** = ***UNCHA** NGED / ***ALL** / ***SYS**TEM / ***SIH**

,**CONNECTION-SCOPE** = ***UNCHA** NGED / ***TASK** / ***PROG**RAM / ***FREE** / ***CALL** / ***OPTIMAL**

,**FIRST-CONNECTION** = ***UNCHA** NGED / ***ALLOWED** / ***FORBIDDEN**

***BY-PROGRAM(...)**

CONNECTION-SCOPE = ***UNCHA** NGED / ***TASK** / ***PROGRAM**

,**REMOVE-SUBS-ENTRIES** = ***NONE** / list-poss(100): <text 1..8>

,**MEMORY-CLASS** = ***UNCHA** NGED / ***SYSTEM-GLOBAL**(...) / ***LOCAL-PRIVILEGED**(...) /

***LOCAL-UNPRIVILEGED**(...) **/ *BY- SLICE**(...)

***SYSTEM-GLOBAL**(...)

SUBSYSTEM**-ACCESS** = ***LOW** / ***SYS**TEM / ***HIGH**

***LOCAL-PRIVILEGED**(...)

SIZE = <integer 1..32767>

***LOCAL-UNPRIVILEGED**(...)

SIZE = ***UNCHA** NGED / <integer 1..32767>

,**SUBSYS**TEM**-ACCESS** = ***UNCHA** NGED / ***LOW** / ***HIGH**

,**START-ADDRESS** = ***UNCHA** NGED / ***ANY** / <x-string 7..8>

***BY- SLICE**(...)

SIZE = **<integer** 1..32767**>**

,**LINK-ENTRY** = ***UNCHA** NGED / <text 1..8 without-sep>(...)

<text 1..8 without-sep>(...)

AUTOLINK = ***ALLOW** ED / ***FORBIDDEN**

,**ADD-REFER-SUBS** = ***NONE** / list-poss(15): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***LOW** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***HIGH** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**MODIFY-REFER-SUBS** = ***NONE** / list-poss(15): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***UNCHA** NGED / ***LOW**EST**-EXIST**ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***UNCHA** NGED / ***HIGH**EST**-EXIST**ING / <c-string 3..8> / <text 3..8>

,**REMOVE-REFER-SUBS** = ***NONE** / list-poss(15): <structured-name 1..8>

,**UNRESOLVED-EXTERNALS** = ***UNCHA** NGED / ***ALLOW**ED / ***FORBIDDEN**

,**CHECK-REFERENCE** = ***UNCHA** NGED / ***Y**ES / ***NO**

,**ADD-RELATED-SUBS** = ***NONE** / list-poss(100): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***LOW** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***HIGH** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**MODIFY-RELATED-SUBS** = ***NONE** / list-poss(100): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***UNCHA** NGED / ***LOW**EST**-EXIST**ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***UNCHA** NGED / ***HIGH**EST**-EXIST**ING / <c-string 3..8> / <text 3..8>

,**REMOVE-RELATED-SUBS** = ***NONE** / list-poss(100): <structured-name 1..8>

Operands

In each case, the default value *UNCHANGED means that the value set in the SET-SUBSYSTEM-ATTRIBUTES statement is to remain valid.
If the type of job entry point declared (MODE operand) or the subsystem-specific address space (MEMORY operand) is changed, all the suboperands of MODE or MEMORY must be assigned a value explicitly, i.e. the operand value *UNCHANGED (default value) will be rejected.

SUBSYSTEM-NAME = <structured-name 1..8>(...)
Specifies the name and version of the subsystem whose attributes are to be changed.

> **VERSION = <c-string> / <text 3..8>**
> The version of the subsystem must be specified in the format “[V][n]n.m[ann]”. The text elements have the following meanings:
>
> nn = Main version (numeric)
> m = Revision version (numeric)
> ann = Update status (a=letter, release status; nn=numeric, correction status)

INSTALLATION-UNIT =
Defines the name of the installed software unit. A value other than *NONE must be specified for all subsystems installed with IMON, and likewise if the value *INSTALLED(LOGICAL-ID=...) was defined for the operands SUBSYSTEM-LIBRARY, REP-FILE, SUBSYSTEM-INFO-FILE, MESSAGE-FILE and SYNTAX-FILE.
The syntax rules described in the “IMON“ manual must be observed when defining the name.

INSTALLATION-UNIT = *NONE
No name is assigned. This entry is not allowed for any subsystems installed with IMON.

INSTALLATION-UNIT = *STD
The name specified with the SUBSYSTEM-NAME operand is used as the new name of the installed software unit.

INSTALLATION-UNIT = <text 1..30>
New name of the installed software unit.

INSTALLATION-USERID =
Specifies a user ID under which the relevant DSSM task expects the subsystem satellites (REP file, object module library, message file, syntax file and subsystem information file) if they have not yet been assigned to a user ID, in other words if the file name was specified without a user ID.

INSTALLATION-USERID = *NONE
The files will not be expected under a specific user ID.

INSTALLATION-USERID = *DEFAULT-USERID
The files will be expected under the default user ID (prefix “$.”) or, if the subsystem is a local subsystem, under the user ID of the calling task.

INSTALLATION-USERID = <name 1..8>
User ID under which the subsystem satellites are to be expected.
If this statement applies to an SSD object, the files will only actually be expected under the user ID specified here if no user ID was specified in the ADD-CATALOG-ENTRY statement (inclusion of the subsystem definitions from the SSD object in the catalog). The ID specified in ADD-CATALOG-ENTRY takes precedence.

COPYRIGHT =
Specifies whether or not a copyright notice is to be displayed when the subsystem is started and, if so, which one.

COPYRIGHT = *NONE
No copyright notice is to be output.

COPYRIGHT = <c-string 1..54>(...)
Text of the copyright notice which is to be output together with the creation date when the subsystem is started.

> **YEAR = *YEAR-1990 / <c-string 4..4>**
> Number of the year which is to appear in the copyright notice as the creation date. This is not subjected to a semantic check.

LIBRARY =
Specifies a new name for the program or object module library (OML) from which the object code for the subsystem is to be loaded when it is activated.

LIBRARY = *STD
When the subsystem is started, the object code will automatically be loaded from the library SYSLNK.<subsysname>.<subsysvers#>. This library is stored under the user ID under which the holder task is running. For local subsystems this is the user ID of the caller, and for global subsystems it is TSOS.
“<subsysvers#>” is a three-character value consisting of the elements “mmm” specified for the operand SUBSYSTEM-NAME=...(VERSION=...).

LIBRARY = *CPLINK
The subsystem which is to be defined is linked to the BS2000 control program (CP) and must have been loaded before DSSM was activated. This operand may be used only in conjunction with the operand CREATION-TIME=*BEFORE-DSSM-LOAD.

LIBRARY = *INSTALLED(...)
The library name must be determined by calling IMON-GPN (administration of installation paths).

If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

> **LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>**
> Logical ID of the program library or object module library.
>
> **DEFAULT-NAME = <filename 1..54 without-gen-vers>**
> Library name if IMON-GPN is not available or if the logical ID is unknown.

LIBRARY = <filename 1..54 without-gen-vers>
Fully qualified file name of the object module library from which the object code for the subsystem is to be loaded.

SUBSYSTEM-LOAD-MODE =
Specifies the load mode of the subsystem (via the BLS-DSSM interface $PBBND1).

SUBSYSTEM-LOAD-MODE = *STD
The BLS (Binder-Loader-Starter system) is called up in STD run mode and loads the subsystem as an object module.

SUBSYSTEM-LOAD-MODE = *ADVANCED
The BLS is called up in ADVANCED run mode and loads the subsystem as a link and load module (LLM).

REP-FILE =
Specifies whether or not system REPs are required for the subsystem which is being defined, and identifies the file in which they are stored. These correction statements are used during activation of the subsystem, and are applied solely to the modules stored and loaded in the object module library, not to other subsystems or the BS2000 control program (CP). A REP file can also be specified for modules of a nonprivileged subsystem.
REP-FILE must not be specified together with LIBRARY=*CPLINK.

REP-FILE = *STD
By default, system REPs are loaded from the file SYSREP.<subsysname>.<subsysvers#>. This library is stored under the user ID under which the holder task is running. For local subsystems this is the user ID of the caller, and for global subsystems it is TSOS.“<subsysvers#>” is a three-character value consisting of the elements “mmm” specified for the operand SUBSYSTEM-NAME=...(VERSION=...).

REP-FILE = *NO
No REP files are to be processed for the subsystem.

REP-FILE = *INSTALLED(...)
The name of the REP file must be determined by calling IMON-GPN (administration of installation paths).
If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

> **LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>**
> Logical ID of the REP file.
>
> **DEFAULT-NAME =**
> Name of the REP file if IMON-GPN is not available or if the logical ID is unknown.
>
> **DEFAULT-NAME = <filename 1..54 without-gen-vers>**
> A new name is assigned.
>
> **DEFAULT-NAME = *NONE**
> No new name is assigned.

REP-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the REP file from which the correction statements are to be read.

REP-FILE-MANDATORY =
Specifies whether or not a REP file declared via the REP-FILE operand is to be processed when loading the subsystem.

REP-FILE-MANDATORY = *NO
The use of a REP file is not mandatory, i.e. neither the REP file nor its entries are to be checked when the subsystem is activated. Even if the REP file cannot be accessed or if individual correction statements are invalid, the subsystem will still be started.

REP-FILE-MANDATORY = *YES
If any of the following errors occurs during processing of the REP file, the attempt to load the subsystem will be terminated:

- the REP file is not cataloged, or it cannot be read
- checking of the correction statements reveals an error
- the name of a correction statement is invalid

- DMS reports an error during access to the NOREF file (this file is used during loading of a subsystem to prevent invalid system REPs from being logged at the operator terminal)

MESSAGE-FILE =
Specifies whether there is a subsystem-specific message file which is to be automatically activated when the subsystem is loaded.

For subsystems which are defined with the creation time AT-DSSM-LOAD, a dependency relation to the MIP subsystem must be specified in the RELATED-SUBSYSTEM operand.

MESSAGE-FILE = *NO
No message file is to be activated. This value is mandatory for all subsystems which are defined with the creation time BEFORE-DSSM-LOAD (see also the CREATION-TIME operand), as it is not possible to activate a message file as early as this.

MESSAGE-FILE = *INSTALLED(...)
The name of the message file must be determined by calling IMON-GPN (administration of installation paths). If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the message file.

DEFAULT-NAME =
Name of the message file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE No new name is assigned.

MESSAGE-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the message file. This will be automatically activated when the subsystem is loaded (START-SUBSYSTEM command) and automatically deactivated when it is unloaded (STOP-SUBSYSTEM).

SUBSYSTEM-INFO-FILE =
Specifies whether or not a subsystem information file (SSINFO) is available. This file contains subsystem-specific data (subsystem satellites and configuration data) which cannot be processed centrally by DSSM.

SUBSYSTEM-INFO-FILE = *NO
No information file is available for the subsystem.

SUBSYSTEM-INFO-FILE = *INSTALLED(...)
The name of the information file must be determined by calling IMON-GPN (administration of installation paths).

If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the information file.

DEFAULT-NAME =
Name of the information file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE A new name is assigned.

SUBSYSTEM-INFO-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the information file. This name is automatically passed to the activation and deactivation routines (INIT-/DEINIT-/CLOSE-CTRL-ROUTINE operands) when they are called.

SYNTAX-FILE =
Specifies whether the subsystem has linked to it a syntax file which will be activated automatically when the subsystem is loaded. For subsystems which are defined with the start attribute MANDATORY-AT-STARTUP, a dependency relation to the SDF subsystem must be declared in the REFERENCED-SUBSYSTEM operand.

SYNTAX-FILE = *NO
No syntax file is to be activated. This value is mandatory for all subsystems which are defined with the start time BEFORE-DSSM-LOAD or AT-DSSM-LOAD (see also the CREATION-TIME operand), as it is not possible to activate a syntax file as early as this.

SYNTAX-FILE = *INSTALLED(...)
The name of the syntax file must be determined by calling IMON-GPN (administration of installation paths).
If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the syntax file.

DEFAULT-NAME =
Name of the syntax file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE
No new name is assigned.

SYNTAX-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the syntax file which is to be automatically activated when the subsystem is loaded.

DYNAMIC-CHECK-ENTRY =
Specifies whether a dynamic identity check is to be carried out on the subsystem. For this purpose, an entry point must be specified, at which both the subsystem name (eight characters) and also the version number (four or seven characters) must be located. DSSM checks whether the identification specified in the definition agrees with the subsystem which is loaded.

DYNAMIC-CHECK-ENTRY = *STD
The entry point specified in the LINK-ENTRY operand is to be applied in carrying out the identity check.

DYNAMIC-CHECK-ENTRY = *NO
No check is to be carried out. However, this value of the operand must not be used for any subsystems which are loaded before DSSM is activated (CREATION-TIME=*BEFORE-DSSM-LOAD).

DYNAMIC-CHECK-ENTRY = <text 1..8 without-sep>
Name of the entry point which is to be used in applying the identity check.

CREATION-TIME =
Specifies the point in time at which activation of the subsystem (CREATE routine) is initiated.
There are two separate phases during system initialization when DSSM takes over the control of system initialization after it has been called by the startup routine:

Phase 1: The DSSM code is loaded, the DSSM task is generated and started.

This task reserves class 5 memory, reads in the subsystem catalog, and starts those subsystems which have been defined with the start attributes BEFORE-DSSM-LOAD and AT-DSSM-LOAD.

After these subsystems have been loaded, control of system initialization returns to the startup routine.

Phase 2: Following a second call, all those subsystems are loaded which have been defined with the start attributes MANDATORY-AT-STARTUP, BEFORE-SYSTEM-READY and AFTER-SYSTEM-READY. In the case of the first two of these start attributes, loading of the subsystems is synchronized with the start routine (i.e. loading must be completed), but for the last of them asynchronous loading is initiated. Control of system initialization returns to the startup routine.

If different versions of a subsystem are to be defined, it is only possible to specify the start attributes, for use in phases 1 and 2 of system initialization, for one of these versions.

CREATION-TIME = *AT-CREATION-REQUEST
The subsystem must be explicitly loaded by means of the START-SUBSYSTEM command.

CREATION-TIME = *AT-SUBSYSTEM-CALL(...)
The subsystem is to be automatically loaded when the first SVC or ISL call is made. This operand value is reserved for subsystems which are called via SVC or ISL.
If two or more versions of a subsystem are defined with this operand value, VERSION-COEXISTENCE=*ALLOWED must be specified for all of these versions and FUNCTION-NUMBER and FUNCTION-VERSION must be specified for their SVC or ISL entry points, which were declared with CONNECTION-ACCESS with a value other than *SIH.
At least one of the specified subsystems must have been declared with SUBSYSTEM-ENTRIES ..., MODE=*SVC or *ISL (corresponding to the value of the ON-ACTION operand).

ON-ACTION =
Determines what initiates automatic loading of the subsystem.

ON-ACTION = *STD
Default setting: loading begins when any SVC entry point belonging to the subsystem is called.

ON-ACTION = *ISL-CALL
Loading begins when any ISL entry point belonging to the subsystem is called.

ON-ACTION = *ANY
Loading begins when any SVC or ISL entry point belonging to the subsystem is called.

CREATION-TIME = *AT-DSSM-LOAD
The subsystem is to be loaded during system initialization (phase 1) under the control of the DSSM task.
The subsystem must be a privileged one, and may only have address or dependency relations to subsystems which are also defined with this start attribute or with the start attribute BEFORE-DSSM-LOAD.
The files for this subsystem must be held on the home pubset under the TSOS user ID, because at the time of startup the user catalog is not accessible and IMPORT-PUBSET processing has not been completed.
It is not permissible to link in a syntax file for these subsystems.

CREATION-TIME = *BEFORE-DSSM-LOAD
The subsystem is to be loaded during system initialization (phase 1), but not under the control of the DSSM task.

Such subsystems are linked to the control program, and do not need to be synchronized with the DSSM task when activated. However, after the subsystem is loaded it again runs under the control of DSSM and can, from the user's point of view, be controlled in the same way as other subsystems.
No address or dependency relations are possible to subsystems which are defined with any other start attribute. Nor is it permissible to link in a message or syntax file. All job entries (SUBSYSTEM-ENTRIES operand) must be declared, because DSSM creates the (privileged) connection to these job entries. Responsibility for ensuring that at least one version of this subsystem is available at any given time rests entirely with the subsystem developer.
The name of the link context for these subsystems must be unique, because DSSM must also honor an unload request even if the subsystem code has not been loaded. An entry point (DYNAMIC-CHECK-ENTRY operand) must have been specified.

CREATION-TIME = *BEFORE-SYSTEM-READY
The subsystem is to be loaded during system initialization (phase 2). Activation is initiated synchronously; not until loading is complete (or a load error occurs) does control return to the startup routine, which can then report "SYSTEM READY".
The subsystem must be privileged, and may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD, AT-DSSM-LOAD or MANDATORY-AT-STARTUP.
The files for this subsystem must be cataloged on the home pubset.
If a nonprivileged subsystem is declared with this operand value, it is automatically assigned the value *AFTER-SYSTEM-READY. SSCM issues a message.

CREATION-TIME = *MANDATORY-AT-STARTUP
The subsystem must be loaded during system initialization (phase 2). Activation is initiated synchronously - as in the case of BEFORE-SYSTEM-READY. By contrast with the latter, however, loading of the subsystem must in this case be completed **successfully**. Otherwise a message is passed to the startup routine reporting that a mandatory subsystem could not be loaded. In this case, the startup routine will decide whether processing should continue or be terminated.
The subsystem must be privileged, and may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD or AT-DSSM-LOAD. The files for this subsystem must be cataloged on the home pubset.

CREATION-TIME = *AFTER-SYSTEM-READY
The loading of this subsystem is to be initiated during system initialization (phase 2). Execution of this routine is not synchronized with the startup routine, which can report "SYSTEM READY" before subsystem loading is complete.

The subsystem may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD, AT-DSSM-LOAD, MANDATORY-AT-STARTUP or BEFORE-SYSTEM-READY. The files for this subsystem must be cataloged on the home pubset.

INIT-ROUTINE =
Specifies whether there is an initialization routine for the subsystem which must be performed when it is started or resumed. In this case, the name of an entry point must have been declared, and DSSM will delegate initialization to the holder task of the subsystem concerned. It is strongly recommended that an entry point be defined for all subsystems which have the start attribute BEFORE-DSSM-LOAD. During loading of the subsystem, (i.e. when the initialization routine is carried out) the subsystem is then informed that DSSM can assume control over the opening and closing of relations.

INIT-ROUTINE = *NO
No initialization routine is to be performed.

INIT-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the initialization routine.
In the holder task, control is passed to the initialization routine, so that the subsystem can initialize itself. To permit this, it is passed:

- the name and the version of the subsystem, as defined in SSMCAT
- the name of the SSINFO file, if one was specified in the SUBSYSTEM-INFO-FILE operand
- the address of the entry point specified during loading and linking (LINK-ENTRY operand)
- the link context name used by the dynamic binder loader (DBL)
- the name of the memory pool (for subsystems in class 5 or class 6 memory), in order that the subsystem can refer to its own selectable units/load units during dynamic loading
- the name of the message file
- the address of the SUBSYSTEM-PARAMETER operand, if a string has been specified in the START-SUBSYSTEM command

At the end of initialization, a return message is expected from the subsystem, indicating whether initialization has been successful and whether the holder task is to be used as a work task (as specified in the ASSIGN-HOLDER-TASK statement). Depending on what is reported here, the task will from then on be controlled by DSSM or by the subsystem.

CLOSE-CTRL-ROUTINE =
Specifies whether the subsystem incorporates a routine for controlling its suspension/deactivation.

If a subsystem is deactivated (by a STOP-SUBSYSTEM or HOLD-SUBSYSTEM command), DSSM passes control to this routine at the identified entry point in the holder task, or (for *DYNAMIC) this is reported via a bourse or FITC link (as determined by return messages during initialization).
The parameters passed are the same ones as are passed for the INIT-ROUTINE operand. Branching to this routine ensures that a connection to the subsystem still exists.

If a CLOSE-CTRL routine exists, it is possible to change versions without interrupting the BS2000 session. At any given point in time, there is exactly one valid version (either the old version is still available, or the new version has already become available). Without a CLOSE-CTRL routine, changing versions always entails interrupting the connection so that the STOPCOM routine of the old version and the INIT routine of the new version can execute (see also "Swapping subsystem versions").

CLOSE-CTRL-ROUTINE = *NO
The subsystem incorporates no routine that controls the deactivation or suspension of the subsystem.

CLOSE-CTRL-ROUTINE = *DYNAMIC
This routine is called up via the bourse or FITC port. The subsystem passes the required parameters to the CLOSE-CTRL routine dynamically at the end of the INIT routine, and DSSM is informed of the identity of the bourse or FITC port.
In order for the CLOSE-CTRL routine to run, an INIT routine (INIT-ROUTINE operand) and a STOPCOM routine (operand STOPCOM-ROUTINE=*NO/*DYNAMIC) must have been specified.
The holder task of the subsystem must be a work task when the CLOSE-CTRL routine is called (ASSIGN-HOLDER-TASK statement).

CLOSE-CTRL-ROUTINE = <text 1..8 without-sep>
Name of the entry point of the relevant subsystem routine.

STOPCOM-ROUTINE =
Specifies whether the subsystem incorporates a routine which can carry out active termination of tasks.

STOPCOM-ROUTINE = *NO
The subsystem concerned incorporates no such routine.

STOPCOM-ROUTINE = *DYNAMIC
This routine is called via the bourse or FITC. The subsystem passes the required parameters to the STOPCOM routine dynamically at the end of the CLOSE-CTRL routine or (if none exists) at the end of the INIT routine. DSSM is informed of the identity of the bourse or FITC port.

A prerequisite for the use of the STOPCOM routine is than an INIT routine has been specified (INIT-ROUTINE operand). When the STOPCOM routine is called, the holder task for the subsystem must be used as a work task (ASSIGN-HOLDER-TASK statement).

STOPCOM-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the subsystem routine concerned.

DEINIT-ROUTINE =
Specifies whether the subsystem incorporates a routine which can carry out deinitialization of the subsystem. This deinitialization routine causes the resources which were requested by the subsystem (memory, files, devices) to be returned.
If a subsystem is deactivated (by a STOP-SUBSYSTEM or HOLD-SUBSYSTEM command), then DSSM passes control to this routine at the identified entry point in the holder task, or (for *DYNAMIC) this is reported via a bourse or FITC link (as determined by return messages during initialization).
If a subsystem is defined with an INIT routine and a CLOSE-CTRL routine, a DEINIT routine – with the same operand value as the CLOSE-CRTL routine – must be specified.
The parameters which are passed are the same as for the INIT-ROUTINE operand. Branching to this routine ensures that calling tasks will no longer be connected to the subsystem and all existing call relations to the subsystem are deleted.

DEINIT-ROUTINE = *NO
The subsystem concerned does not incorporate a deinitialization routine to request the release of resources; this is done by DSSM itself.

DEINIT-ROUTINE = *DYNAMIC
The routine is called via the bourse or FITC.
The subsystem passes the required parameters to the DEINIT routine dynamically at the end of the STOPCOM routine or, if none exists, at the end of the CLOSE-CTRL routine or, if neither a STOPCOM nor a CLOSE-CTRL routine is incorporated, at the end of the INIT routine. DSSM is informed of the identity of the bourse or FITC port.
A prerequisite for the use of the DEINIT routine is than an INIT routine has been specified (INIT-ROUTINE operand). When the DEINIT routine is called, the holder task for the subsystem must be used as a work task (ASSIGN-HOLDER-TASK statement).

DEINIT-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the subsystem routine concerned.

STOP-AT-SHUTDOWN =
Specifies whether the subsystem is to be unloaded automatically at shutdown after the user tasks have terminated.

STOP-AT-SHUTDOWN = *NO
The subsystem will not be unloaded automatically.
This parameter should not be specified for subsystems which have address relations to other subsystems which are defined with STOP-AT-SHUTDOWN=*YES.

STOP-AT-SHUTDOWN = *YES
The subsystem will be unloaded automatically at shutdown.
This specification will be ignored if no STOPCOM, DEINIT or CLOSE-CRTL routine is specified. In this case, SSCM issues a message.

INTERFACE-VERSION =
Identifies the entry point via which DSSM can access the interface version which is to be used for calling the INIT, DEINIT, STOPCOM or CLOSE-CTRL routine.

INTERFACE-VERSION = *NO
The subsystem does not call a INIT, DEINIT, STOPCOM or CLOSE-CTRL routine.

INTERFACE-VERSION = <text 1..8 without-sep>
Name of the entry point. The entry point points to the standard header where the interface version is stored. The standard header is generated by calling the macro $ESMINT(I) with MF=I/L.
This operand is mandatory for subsystems for which an INIT, DEINIT, STOPCOM or CLOSE-CTRL routine has been specified.

SUBSYSTEM-HOLD =
Specifies whether the subsystem which is loaded may be suspended or unloaded.

SUBSYSTEM-HOLD = *ALLOWED
The subsystem which is loaded may be suspended and unloaded. The commands HOLD-SUBSYSTEM and STOP-SUBSYSTEM are permissible for this subsystem.

SUBSYSTEM-HOLD = *FORBIDDEN
The commands HOLD-SUBSYSTEM and STOP-SUBSYSTEM must not be used for this subsystem; it will only be unloaded at shutdown - as specified by the STOP-AT-SHUTDOWN operand.
Unloading the subsystem by replacing it with another subsystem entails no interruption.

STATE-CHANGE-CMDS =
Specifies whether or not the DSSM commands for controlling the subsystem (START-SUBSYSTEM, STOP-SUBSYSTEM, HOLD-SUBSYSTEM and RESUME-SUBSYSTEM) may be used during a session.

If a changeover is made from one version of a subsystem to another, the value specified for STATE-CHANGE-CMDS for the version being replaced is ignored.

STATE-CHANGE-CMDS = *ALLOWED
The commands may be used from the operator terminal and under the privileged user ID (the user ID which has the SUBSYSTEM-MANAGEMENT system privileges).

STATE-CHANGE-CMDS = *FORBIDDEN
The commands must not be used - neither from the operator terminal nor under the privileged user ID.

STATE-CHANGE-CMDS = *BY-ADMINISTRATOR-ONLY
The commands may only be used under the privileged user ID; the commands are not available to the operator at the operator terminal.
If a subsystem is deactivated (with a STOP-SUBSYSTEM or HOLD-SUBSYSTEM command), DSSM passes control to this routine at the specified entry point in the holder task, or (if *DYNAMIC was specified) this is reported via a bourse or FITC link (as determined by return messages during initialization).
The parameters passed are the same ones as are passed for the INIT-ROUTINE operand. Branching to this routine ensures that calling tasks will no longer be connected to the subsystem. Tasks which are still in a call relation to the subsystem remain unaffected by this.

FORCED-STATE-CHANGE =
Specifies whether use of the operand FORCED=*YES is permitted within the commands STOP-SUBSYSTEM and HOLD-SUBSYSTEM. This function can be used to force the unconditional deactivation of the subsystem.

FORCED-STATE-CHANGE = *FORBIDDEN
It is not possible to force deactivation of the subsystem. DSSM will reject any use of the FORCED operand in the commands concerned, and will issue a corresponding error message.

FORCED-STATE-CHANGE =* ALLOWED
The operand FORCED=*YES may be used for this subsystem.
This operand value must not be used in conjunction with SUBSYSTEM-HOLD=*FORBIDDEN.

RESET =
Specifies whether the operand RESET=*YES is permitted within the commands START-SUBSYSTEM and RESUME-SUBSYSTEM. This function can be used to force the unconditional loading or resumption of the subsystem, even if the state of the subsystem is currently IN-DELETE or IN-HOLD.

RESET = *FORBIDDEN
It is not possible to force the activation of the subsystem. DSSM will reject the use of the RESET operand in the commands concerned, and will issue a corresponding error message.

RESET = *ALLOWED
The operand RESET=*YES may be used for this subsystem.
This operand value must not be specified together with SUBSYSTEM-HOLD=*FORBIDDEN.

RESTART-REQUIRED =
Specifies whether the initialization routine for the subsystem is to be executed if the holder task terminates abnormally.

RESTART-REQUIRED = *NO
The initialization routine is not used to restart the subsystem.

RESTART-REQUIRED = *YES
If the holder task terminates abnormally, the initialization routine should be used. Provision must have been made in the INIT-ROUTINE operand for executing this routine.

VERSION-COEXISTENCE =
Specifies whether more than one version of the same subsystem may be active at a time.

VERSION-COEXISTENCE = *FORBIDDEN
The current version of the subsystem cannot coexist with another version of the same subsystem.

VERSION-COEXISTENCE = *ALLOWED
The current version of the subsystem can coexist with another version of the same subsystem (coexistence mode). In the definition of the job entry point (SUBSYSTEM-ENTRIES operand), indirect links via system exit routines must not have been specified. If different versions of the same subsystem are loaded and the same job entry point is defined for these, the link which is implemented is always to the highest loaded version of the subsystem.
If coexistent subsystems access coexistent syntax files, the latter must have been declared in the SSD object and cannot be administered by SDF.
However, where the links are via SVC and ISL, it is possible to select a version using the operands FUNCTION-NUMBER and FUNCTION-VERSION.

VERSION-EXCHANGE =
Specifies whether a subsystem may be loaded in exchange mode. Exchange mode allows the temporary coexistence of two versions of the same subsystem. If version B of a subsystem is loaded whilst version A of the subsystem is already active, all new callers will be connected to version B. Jobs which are connected to version A will still be processed. When all the jobs which use version A have been processed, this will automatically be terminated.

In the definition it should be noted that the “old” version which is being replaced must not be dependent on the “new” version which replaces it.

VERSION-EXCHANGE = *FORBIDDEN
The current version of the subsystem must not be replaced.

VERSION-EXCHANGE = *ALLOWED
Exchange mode, which allows the temporary coexistence of two subsystems, is permitted for the current subsystem version.

ADD-SUBS-ENTRIES / MODIFY-SUBS-ENTRIES =
Indicates whether new job entries are to be defined (ADD), or the attributes of existing job entries are to be changed (MODIFY).

ADD-SUBS-ENTRIES / MODIFY-SUBS-ENTRIES = *NONE
Default value: new job entries are not to be added, nor are the attributes of existing job entries to be modified.

ADD-SUBS-ENTRIES / MODIFY-SUBS-ENTRIES = list-poss(100): <text 1..8>
Either declares names of entry points for a maximum of 100 new job entries for the subsystem (for each of which the type must be defined in the substructures (ADD), or modifies job entry points which have already been defined (MODIFY).

MODE =
Defines the type of a job entry point which is defined for the subsystem.
If the type of the entry point declared is modified, all the suboperands of MODE must be assigned a value explicitly; i.e. the operand value *UNCHANGED (default value for MODIFY-SUBS-ENTRIES) will be rejected.

MODE = *LINK
The job entry point cannot be accessed by indirect linkage, but only by using a CONNECT relation through an external linkage editor symbol.
In the case of different versions of the same subsystem which use the same external linkage editor symbol, DSSM automatically sets up the link to the highest loaded version of the subsystem.

MODE = *ISL(...)
The job entry is effected by indirect linkage via System Procedure Linkage (for privileged subsystems only). If the specification includes in addition a function and version number for the ISL entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem (if VERSION-COEXISTENCE=*ALLOWED is specified).

For different subsystems, if the job entry point is to be accessed by the same ISL entry point, they must be uniquely identified by specifying the function and version numbers.In the case of different versions of the same subsystem which use the same ISL entry point, then - if the function and version numbers are not specified - DSSM will automatically set up a connection to the highest loaded version of the subsystem.
In the case of different versions of the same subsystem which use the same ISL entry point and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller’s parameter list.
It is not permissible to enter a value of *ALL for the CONNECTION-ACCESS operand in reference to ISL entry points.

FUNCTION-NUMBER =
Specifies whether a particular function and version number of the ISL entry point is to be addressed, as the same ISL entry point can be used by different functions.

FUNCTION-NUMBER = *NONE
Default value: no particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)
Number of the ISL entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>
Version of the specified ISL function number.

MODE = *SVC(...)
Job entry is to be effected by an indirect connection using a supervisor call (SVC).

If the specification includes in addition a function and version number for the SVC entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem (if VERSION-COEXISTENCE=*ALLOWED is specified).
For different subsystems, if the job entry is to be accessed by the same SVC, they must be uniquely identified by specifying the function and version numbers.
In the case of different versions of the same subsystem which use the same SVC, then – if the function and version numbers are not specified – DSSM will automatically set up a connection to the highest loaded version of the subsystem.
In the case of different versions of the same subsystem which use the same SVC and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller’s parameter list.

NUMBER = <integer 0..255>
Number of the SVC via which job entry is to be effected. No SVC number greater than 191 may be used in conjunction with CONNECTION-ACCESS=*ALL.

CALL-BY-SYSTEM-EXIT =
Defines whether the specified SVC number may be called from within system exit routines.

CALL-BY-SYSTEM-EXIT = *ALLOWED
System exit routines are permitted to call the specified SVC number.

CALL-BY-SYSTEM-EXIT = *FORBIDDEN
System exit routines are not permitted to call the specified SVC number.

FUNCTION-NUMBER =
Specifies whether a particular function and version number of the SVC entry point is to be addressed, as the same SVC entry point can be used by different functions.

FUNCTION-NUMBER = *NONE
No particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)
Number of an SVC entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>
Version of the specified SVC function number.

MODE = SYSTEM-EXIT(...)
Job entry is to be effected by an indirect connection using system exit routines.

This operand must not be used in conjunction with CONNECTION-ACCESS=*ALL.

NUMBER = <integer 0..127>
Number of the system exit routine.

CONNECTION-ACCESS =
Specifies the access authorization (privileges) required by the subsystem.

CONNECTION-ACCESS = *ALL
Privileged and nonprivileged program runs may access the subsystem.
This operand value must not be used in conjunction with MODE=*SYSTEM-EXIT/*ISL/*SVC (with an SVC number greater than 191).

CONNECTION-ACCESS = *SYSTEM
Only privileged program runs may access the subsystem.

CONNECTION-ACCESS = *SIH
Only tasks running in the SIH processor state may access the subsystem.
The subsystem called also runs in the SIH processor state, i.e. it is uninterruptible.This operand value is permissible only for subsystems for which the entry point is defined via:

- System Procedure Linkage (MODE=*ISL(FUNCTION-NUMBER=*NONE))
- CONNECTION-SCOPE=*OPTIMAL
- MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*SYSTEM)

CONNECTION-SCOPE =
Identifies the event which will call up the automatic cleardown of the connection to the specified subsystem job entry.

CONNECTION-SCOPE = *TASK
The connection will be cleared when the task terminates.

CONNECTION-SCOPE = *PROGRAM
The connection will be cleared when the program terminates, or before.
Only CONNECTION-SCOPE=*PROGRAM may be specified in conjunction with MEMORY-CLASS=*LOCAL-UNPRIVILEGED.
This operand value is recommended for subsystems which were declared with SUBSYSTEM-ACCESS=*LOW/*HIGH or MEMORY-CLASS=*BY-SLICE.

CONNECTION-SCOPE = *FREE
DSSM is not to carry out any checking of the connections to the job entry point. The connection will not be automatically cleared - unless explicitly requested. To avoid problems or possible errors when the subsystem is being unloaded, the connections must be managed by the subsystem itself.

CONNECTION-SCOPE = *CALL
On return from this job entry point, DSSM will automatically clear the connections.This operand value is only available with subsystems for which the job entry point is defined by means of System Procedure Linkage (ISL) or supervisor calls (SVC).

CONNECTION-SCOPE = *OPTIMAL
The subsystem is deactivated or suspended when there are no further tasks with a connection to this entry point.
A routine with an entry point defined with *OPTIMAL must be terminated with RETURN.If an entry point of a subsystem is defined with CONNECTION-SCOPE=*OPTIMAL, all of its entry points must be defined in the subsystem catalog with MODE!=*LINK.
While a subsystem is being deactivated or suspended, no call of the subsystem with CONNECTION-SCOPE=*OPTIMAL is accepted.

FIRST-CONNECTION =
Determines whether or not first connection of the task to the specified job entry point in the subsystem is allowed. At least one job entry point of a subsystem must be defined with FIRST-CONNECTION=*ALLOWED.

FIRST-CONNECTION = *ALLOWED
First connection to the specified job entry point is allowed.
This value is the default setting for the ADD-SUBS-ENTRIES statement.

FIRST-CONNECTION = *FORBIDDEN
Connection to the specified job entry point via SVC or ISL is not allowed if the task has not yet been connected to another job entry point belonging to the subsystem.
It is not permitted to specify this operand value for job entry points that have been defined with MODE=*LINK/*SYSTEM-EXIT or CONNECTION-ACCESS=*SIH.

MODIFY-SUBS-ENTRIES = *NONE / list-poss(100): <text 1..8> / *BY-PROGRAM(...)
The values *NONE and list-poss(100): <text 1..8> are described at the ADD-SUBS-ENTRIES operand.

MODIFY-SUBS-ENTRIES = *BY-PROGRAM(...)
The entry points of the specified subsystem are supplied dynamically from the BLS name list at load time instead of statically from the catalog. A prerequisite for this functionality is the use of BLSSERV as of version 2.1, that supports using EEN names as entry points for DSSM subsystems.

The statement is rejected if the subsystem was not previously also defined with *BY-PROGRAM with the SET-SUBSYSTEM-ATTRIBUTES statement. MODIFY-SUBS-ENTRIES is used for changing the connection settings.

If *BY-PROGRAM is used, the ADD-SUBS-ENTRIES and REMOVE-SUBS-ENTRIES operands must be specified with *NONE.

CONNECTION-SCOPE = *TASK / *PROGRAM
The connection is shut down at task or program termination.

REMOVE-SUBS-ENTRIES =
Defines whether or not existing job entries which have been defined for the subsystem are to be deleted.

REMOVE-SUBS-ENTRIES = *NONE
Default value: none of the job entries is to be deleted.

REMOVE-SUBS-ENTRIES = list-poss(100): <text 1..8>
Names of the job entries (up to 100) which are no longer to apply to the subsystem.

MEMORY-CLASS =
Specifies the subsystem-specific address space in which the subsystem is to be loaded. System administration can use this operand to define the address space valid for the subsystem concerned so as to meet the special requirements of the installation.
If the subsystem-specific address space is modified, each of the suboperands of MEMORY must be assigned a value explicitly (the operand value *UNCHANGED (default value) will be rejected).

MEMORY-CLASS = *SYSTEM-GLOBAL(...)
The subsystem will be loaded in class 3 or class 4 memory. Resident CSECTs will be given class 3 memory, all others will be given pageable class 4 memory.

SUBSYSTEM-ACCESS =
Identifies the access authorization (privileges) and the location of the requested memory.

SUBSYSTEM-ACCESS = *LOW
Nonprivileged address space below the 16-Mbyte boundary is allocated.

SUBSYSTEM-ACCESS = *SYSTEM
Subsystems declared with this operand value are privileged subsystems to which privileged address space above the 16-Mbyte boundary is allocated.

This operand value is mandatory for subsystems whose entry point is declared via SVC (MODE=*SVC) or for which an INIT, STOPCOM, DEINIT or CLOSE-CTRL routine is declared. It is not permitted in combination with CONNECTION-ACCESS=*ALL and MODE=*LINK.

SUBSYSTEM-ACCESS = *HIGH
Nonprivileged address space up to 2 Gbytes is allocated.

MEMORY-CLASS = *LOCAL-PRIVILEGED(...)
The subsystem is given a memory pool in nonprivileged class 5 memory, located below the 16-Mbyte boundary. This specification is suitable for nonprivileged subsystems which demand a relatively large amount of address space (approx. 1 Mbyte) and have to be set up below the 16-Mbyte boundary. These subsystems are loaded in memory pools at the same address, in order to manage the use of the limited address space below 16 Mbytes. Although such subsystems are loaded in parallel in the same address space, they cannot be used simultaneously by the same task (see also the SEPARATE-ADDRESS-SPACE statement).
The subsystem must not contain any resident CSECTs as otherwise a later attempt to activate it will be aborted.

SIZE = <integer 1..32767>
Size of the required address space (in 4Kbyte pages) for the memory pool in class 5 memory. This value should be set at least high enough to ensure that the subsystem and any selectable units/load units which it may load dynamically can be loaded in their entirety. The upper limit is generation-specific.

MEMORY-CLASS = *LOCAL-UNPRIVILEGED(...)
The subsystem is given a memory pool in nonprivileged class 6 memory. This specification is reserved for subsystems which can be executed like a program.
In keeping with this, their access authorization (privileges) must be defined with the value *ALL in the CONNECTION-ACCESS operand.
This operand value must not be specified together with an entry point which was defined with CONNECTION-ACCESS=*SYSTEM.
The subsystem must not contain any resident CSECTs as otherwise a later attempt to activate it will be aborted.
If this operand value is specified, only CONNECTION-SCOPE=*PROGRAM is permissible for clearing the connection to the specified subsystem entry point.

SIZE = <integer 1..32767>
Size of the required address space (in 4Kbyte pages) for the memory pool in class 6 memory. This value should be set at least high enough to ensure that the subsystem and any selectable units/load units which it may load dynamically can be loaded in their entirety. The upper limit is generation-specific.

SUBSYSTEM-ACCESS =
Identifies the location of the requested memory space.

SUBSYSTEM-ACCESS = *LOW
Nonprivileged address space below the 16-Mbyte boundary is allocated.
Because this specification is suitable for subsystems which can be executed like programs, it is advisable additionally to specify CONNECTION-SCOPE=*PROGRAM.

SUBSYSTEM-ACCESS = *HIGH
Nonprivileged address space up to 2 Gbytes is allocated.
Because this specification is suitable for subsystems which can be executed like programs, it is advisable additionally to specify CONNECTION-SCOPE=*PROGRAM.

START-ADDRESS =
Defines the start address in class 6 memory.

START-ADDRESS = *ANY
The location of the subsystem in class 6 memory will be determined by DSSM.

START-ADDRESS = <x-string 7..8>
Start address in the segment raster at which the subsystem’s start address is to be located. The value specified must be an 8-character hexadecimal constant which is a multiple of X'100000'.

MEMORY-CLASS = *BY-SLICE(...)
The specified subsystem is a nonprivileged subsystem and consists of an LLM, which in turn consists of a shareable code (program area) and a non-shareable code (data area). The program area is loaded into the shareable address space (this corresponds to MEMORY-CLASS=*SYSTEM-GLOBAL). The data area is loaded into the user address space of the holder task and is copied into the private user address spaces of the connected tasks at the same address.
The following values must be specified together with MEMORY-CLASS=*BY-SLICE: SUBSYSTEM-LOAD-MODE=*ADVANCED and CONNECTION-ACCESS=*ALL.

SIZE = <integer 1..32767>
Specifies the size of the requested memory space for the data area in 4K pages. The value chosen here must be sufficiently large to allow the subsystem and, if appropriate, selectable units/load units dynamically loaded by the subsystem to be loaded in full. The upper limit is dependent on generation.

LINK-ENTRY = <text 1..8 without-sep>(...)
Defines the name of the object module/ENTRY/CSECT required for loading (for the operand in the call of the $PBBND1 macro to the dynamic binder loader DBL). The subsystem must be loaded in its entirety by this ENTRY (if necessary, using autolink).

AUTOLINK =
Controls invocation of the autolink function during linking and loading.
The linkage editor’s autolink function permits the automatic insertion of modules which are not explicitly inserted by appropriate statements. The main purpose of this function is to save users of higher-level programming languages from having to make explicit statements to insert individually the numerous modules of the runtime system which are required. Further details of the autolink function can be found in the “BLSSERV” manual.
The autolink function can also be implicitly circumvented if the first external reference encountered during linkage editing of the object module which is to be loaded points to a prelinked module. The advantage of this approach is that the paging behavior when the subsystem is later executed can be optimized at this preliminary stage (during linkage editing). In addition, errors during linkage editing can be avoided in this way.

AUTOLINK = *ALLOWED
The autolink function is allowed.

AUTOLINK = *FORBIDDEN
The autolink function is suppressed.

ADD-REFER-SUBS / MODIFY-REFER-SUBS =
Specifies whether to set up a list containing subsystems to which there are address relations, for use in resolving external references (ADD), or if there already exists a list which is to be modified (MODIFY).

ADD-REFER-SUBS / MODIFY-REFER-SUBS = *NONE
Default value: no list is to be set up, nor is an existing list to be modified.

ADD-REFER-SUBS / MODIFY-REFER-SUBS = list-poss(15):<structured-name 1..8>
There are external references to be specified (ADD) or modified (MODIFY).
External references can be nominated for a maximum of 15 other subsystems; these subsystems must be used in resolving the external references. If any of the subsystems nominated here is missing at the time of activation or deactivation (and if a check of the external references has also been requested by the operand CHECK-REFERENCE=*YES), the action will be aborted.
It is also possible to address the BS2000 control program via these external references - using the name CP. In the substructure which follows, it is possible to specify either exactly one version, or a range of versions within which all versions are to be referred to.
If a version range list is used to limit the version of the CP subsystem, DSSM checks the compatibility of the current CP version against the versions in the range list. The subsystem will only be loaded if it is a compatible version.

The following restrictions should be noted when specifying subsystems to which there are address relations:

- No address relations may be declared to subsystems which have the attribute MEMORY-CLASS=*LOCAL-PRIVILEGED/*LOCAL-UNPRIVILEGED/*BY-SLICE.
- If the attribute SUBSYSTEM-ACCESS=*SYSTEM is specified for the subsystem which is being defined, no subsystem may be addressed if it is defined with SUBSYSTEM-ACCESS=*LOW or SUBSYSTEM-ACCESS=*HIGH.
- Subsystems which have the attribute STOP-AT-SHUTDOWN=*YES may have address relations only to other subsystems which also have this attribute.
- As a rule, a nonprivileged subsystem must not have any address relations to the control program (CP)
- If a reference is made to a subsystem which has at least one version that may be operated in coexistence or exchange mode, a unique version must be specified
- Any address relations must be defined in accordance with the start attributes (CREATION-TIME operand); i.e. a subsystem may have relations to other subsystems only if these were started at the same time or earlier.

LOWEST-VERSION =
Specifies the lowest value (lowest version) in the subsystem version range list.

LOWEST-VERSION = *LOWEST-EXISTING
The lowest version in the catalog is to be addressed.

LOWEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the lower limit of the range of versions.

HIGHEST-VERSION =
Specifies the upper value (highest version) in the subsystem version range list.

HIGHEST-VERSION = *HIGHEST-EXISTING
The highest version in the catalog is to be addressed.

HIGHEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the upper limit of the range of versions.

MODIFY-REFER-SUBS =
See the description of ADD-REFER-SUBS.

REMOVE-REFER-SUBS =
Indicates whether or not existing external references to other subsystems are to be deleted.

REMOVE-REFER-SUBS = *NONE
Default value: none of the existing external references to other subsystems is to be deleted.

REMOVE-REFER-SUBS = list-poss(15): <structured-name 1..8>
Names of a maximum of 15 external references which are to be made invalid.

UNRESOLVED-EXTERNALS =
Defines how the load procedure is to behave if there are unresolved external references.

UNRESOLVED-EXTERNALS = *FORBIDDEN
If unresolved external references occur, the load procedure will be terminated.

UNRESOLVED-EXTERNALS = *ALLOWED
The load procedure will be continued; unresolved external references will be given the value X'FFFFFFFF'.

CHECK-REFERENCE =
Defines whether or not the subsystems specified in the REFERENCED-SUBSYSTEM operand are to be checked in respect of their status and availability.

CHECK-REFERENCE = *YES
The referenced subsystems will be checked. If any of them is missing, DSSM will abandon the activation or unloading of the subsystem.

CHECK-REFERENCE = *NO
DSSM is not to carry out any check.
Even if the user generates complex subsystems with this statement, DSSM will still execute the requested functions **despite** the risk of conflicts:

- The START-SUBSYSTEM command loads the specified subsystem even if a subsystem to which defined relations exist has not yet been fully loaded.
- DSSM executes the RESUME-SUBSYSTEM, STOP-SUBSYSTEM and HOLD-SUBSYSTEM commands without performing any check on relations or dependencies.

ADD-RELATED-SUBS / MODIFY-RELATED-SUBS =
Specifies whether a list of subsystems for which dependency relations exist are to be created (ADD) or an existing list of dependent subsystems is to be modified (MODIFY).

ADD-RELATED-SUBS / MODIFY-RELATED-SUBS = *NONE
Default value: no dependency relations are to be defined or modified for the subsystem.

ADD-RELATED-SUBS / MODIFY-RELATED-SUBS = list-poss(100): <structured-name 1..8>
Dependency relations are to be defined (ADD) or modified (MODIFY) for up to 100 other subsystems, without which the subsystem currently being defined cannot function.
It is also permissible to define dependency relations to the BS2000 control program (CP). The rules and restrictions which apply when doing so are analogous to those for address relations, where they are described in more detail (see the ADD-REFER-SUBS operand).A dependency relation always points to a single version of a subsystem. In the substructure which follows, it is possible to specify either exactly one version, or a range of versions within which all versions are to be referred to.

The following general restrictions should be noted when specifying dependent subsystems:

- The relation which is defined must not contain closed loops. A loop arises if subsystem A is dependent on B, B is dependent on C, and this is in turn dependent on A

- If the subsystem which is being defined has been given the attribute MEMORY-CLASS=*SYSTEM-GLOBAL, then it is not permissible to address any subsystems defined with MEMORY-CLASS=*LOCAL-PRIVILEGED or *LOCAL-UNPRIVILEGED.
- For subsystems which have the attribute SUBSYSTEM-ACCESS=*SYSTEM, no dependency relations may be defined to subsystems to which SUBSYSTEM-ACCESS=*LOW or SUBSYSTEM-ACCESS=*HIGH or MEMORY-CLASS=*BY-SLICE applies.
- The dependency relations must be defined to correspond to the start attributes (CREATION-TIME operand); i.e. a subsystem may only be dependent on subsystems which are started at the same time or earlier.

LOWEST-VERSION =
Specifies the lowest value (lowest version) in the subsystem version range list.

LOWEST-VERSION = *LOWEST-EXISTING
The lowest version in the catalog is to be addressed.

LOWEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the lower limit of the range of versions.

HIGHEST-VERSION =
Specifies the upper value (highest version) in the subsystem version range list.

HIGHEST-VERSION = *HIGHEST-EXISTING
The highest version in the catalog is to be addressed.

HIGHEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the upper limit of the range of versions.

MODIFY-RELATED-SUBS =
See the description of ADD-RELATED-SUBS.

REMOVE-RELATED-SUBS =
Specifies whether existing dependency relations to other subsystems are to be deleted.

REMOVE-RELATED-SUBS = *NONE
Default value: none of the existing dependency relations to other subsystems is to be deleted.

REMOVE-RELATED-SUBS = list-poss(100): <structured-name 1..8>
Names of a maximum of 100 subsystems to which all dependency relations are to be removed.

4.3.7 MODIFY-WORK-TASK-ATTRIBUTE

Modify work task parameters

Function

This statement can be used to specify subsystems which are no longer to use their holder tasks as a work task. It is possible to specify in addition a new TSN which is to be given to the holder task. This statement will be rejected if the specified subsystem does not exist, or if it does not use the holder task as a work task.

MODIFY-WORK-TASK-ATTRIBUTE will be rejected if neither of the following statements has been executed beforehand:

- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

Format

MODIFY-WORK-TASK-ATTRIBUTE
SUBSYSTEM-NAME = <structured-name 1..8> ,**SUBSYS**TEM**-VERS**ION = <c-string 3..8> / <text 3..8> ,**WORK-TASK** = ***UNCHA**NGED (...) / ***NO** ***UNCHA**NGED(...) **TSN** = ***BY-DSSM** / <alphanum-name 1..4>

Operands

SUBSYSTEM-NAME = <structured-name 1..8>
Name of the subsystem which is using the holder task as a work task.

SUBSYSTEM-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is using the holder task as a work task.
This version must have been declared previously.

WORK-TASK =
Specifies whether the holder task is no longer to be used as a work task, or if the holder task is simply to be given a new TSN.

WORK-TASK = *UNCHANGED
The subsystem is to continue to use the holder task as a work task. However, a new TSN can be issued for the holder task, in the substructure which follows.

TSN =
Specifies the task sequence number (TSN) to be given to the subsystem’s work task.

TSN = *BY-DSSM
Default value: the TSN is issued by DSSM when the work task is loaded.

TSN = <alphanum-name 1..4>

Task sequence number to be given to the work task when it is started.

The specified TSN must be uniquely defined and usable when the subsystem is loaded.

WORK-TASK = *NO

The holder task for the specified subsystem is no longer to be used as a work task.

4.3.8 REMOVE-ADDR-SPACE-SEPARATION

Revoke disjunctive distribution of subsystems in class 5 memory

Function

This statement can be used to revoke the declarations made using a SEPARATE-ADDRESS-SPACE statement and specifying the non-overlapping (disjunctive) distribution of subsystems in class 5 memory.

Use of this statement is irrelevant for subsystems in class 3 or class 4 memory (operand MEMORY-CLASS=*SYSTEM-GLOBAL in the SET-SUBSYSTEM-ATTRIBUTES statement) and for subsystems in class 6 memory (MEMORY-CLASS=*LOCAL-UNPRIVILEGED).
Subsystems in class 6 memory are never used in parallel, and may therefore overlap in the address space; subsystems in class 3 or 4 memory never overlap.

REMOVE-ADDR-SPACE-SEPARATION will be rejected if neither of the following statements has been executed beforehand:

- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

The name of the subsystem must be contained in the catalog. If any of the address relations specified in the list of subsystems does not exist, it will be ignored; processing of the statement will continue.

Format

REMOVE-ADDR-SPACE-SEPARATION
SUBSYSTEM-NAME = <structured-name 1..8> ,**FROM-SUBSYSTEMS** = list-poss(15): <structured-name 1..8>

Operands

SUBSYSTEM-NAME = <structured-name 1..8>
Name of the subsystem which until now was not permitted to overlap with other subsystems.

The subsystem must be known to SSCM and must already be defined.

FROM-SUBSYSTEMS = list-poss(15): <structured-name 1..8>
List of a maximum of 15 subsystems for each of which a declaration has been made to the effect that it must not overlap with the subsystem named in the SUBSYSTEM-NAME operand. The declaration will be revoked for these subsystems.

4.3.9 REMOVE-CATALOG-ENTRY

Logically delete definition of subsystem from subsystem catalog

Function

This statement is used to logically delete the definition of a subsystem which is stored in the subsystem catalog.

The statement will be rejected if a current (non-empty) catalog, in which the definition is stored, has not been opened by a START-CATALOG-CREATION or START-CATALOG-MODIFICATION statement.

The statement will also be rejected if the subsystem specified is not contained in the catalog named. If the subsystem is removed from the catalog, then any disjunctive relations (declared by a SEPARATE-ADDRESS-SPACE statement) will also be deleted.

> ¡ Where there are dependency or load relations, the removal of a subsystem may lead to errors during consistency checks on the catalog.

Format

REMOVE-CATALOG-ENTRY
SUBSYSTEM-NAME = <structured-name 1..8>(...)   <structured-name>(...)   **VERSION** = ***ALL** / <c-string 3..8> / <text 3..8>

Operands

SUBSYSTEM-NAME = <structured-name 1..8>(...)
Name of the subsystem whose definition is to be deleted from the catalog.

VERSION = *ALL / <c-string 3..8> / <text 3..8>
Specifies the version of the subsystem whose definition is to be deleted from the catalog. If VERSION=*ALL is specified, the definitions of all versions of the specified subsystem are deleted from the catalog.

4.3.10 SAVE-CATALOG

Save subsystem catalog as PAM file

Function

With the aid of this statement, a subsystem catalog created in memory by means of statements entered earlier can be saved as a PAM file.

The file name is either derived from the last START-CATALOG-CREATION statement or from the START-CATALOG-MODIFICATION statement , or a new file name can be specified in fully qualified form.

SAVE-CATALOG will be rejected with an error message:

- if neither of the statements START-CATALOG-CREATION or START-CATALOG-MODIFICATION has been executed beforehand
- if the catalog created in memory contains no definitions, i.e. is empty
- if the catalog already contains a file with the name (other than *CURRENT) specified in CATALOG-NAME and REPLACE-OLD-FILE=*NO was specified.

Note that this statement must be preceded by a check run for the catalog. This check run can be initiated by a CHECK-CATALOG statement .

Format

SAVE-CATALOG
CATALOG-NAME = ***CURR** ENT / <filename 1..51 without-gen-vers>(...) **REPLACE-OLD-FILE** = ***NO** / ***Y**ES ,**FORCED** = ***NO** / ***Y**ES / ***FOR-ADD-SUBSYSTEM**

Operands

CATALOG-NAME =
File name under which the catalog is to be saved.

CATALOG-NAME = *CURRENT
The catalog will be saved under the file name specified in the last START-CATALOG-CREATION or START-CATALOG-MODIFICATION statement.

CATALOG-NAME = <filename 1..51 without-gen-vers>
Fully qualified file name under which the catalog is to be saved.

REPLACE-OLD-FILE= *NO / *YES
Specifies whether an existing, cataloged file with the name specified in CATALOG-NAME may be overwritten (*YES) or not (*NO).

FORCED =
Determines how SSCM is to behave if there are errors in the subsystem definition.

FORCED = *NO

Default value: in the case of errors which SSCM recognizes during the analysis of the subsystem definition, the catalog is not to be saved in a PAM file.
Errors may arise in the definitions of subsystems due to a set of relations (address, dependency or external reference relations) which contains loops. Such a loop results if a relation (“ --> ”) is defined which, after a number of steps, refers back to itself:

Subsystem A --> Subsystem B
Subsystem B --> Subsystem C
Subsystem C --> Subsystem A

FORCED = *YES

In spite of any errors which SSCM may identify during the analysis of the subsystem definitions, the catalog is to be saved in a PAM file.
Responsibility for a catalog saved in this way lies with the user; the behavior of DSSM cannot be guaranteed.

FORCED = *FOR-ADD-SUBSYSTEM

The specified catalog will be saved even if the following errors arise:

- a subsystem includes dependency relations to undefined subsystems or
- a subsystem is to share a holder task with an undefined subsystem.

Use of this operand is a good idea when a catalog is to be created that contains nothing but new subsystems, which may, however, have relations to other subsystems defined in the old catalog. This new catalog is added to the existing old catalog by means of the command ADD-SUBSYSTEM ...,TYPE=*NEW-SUBSYSTEMS.

4.3.11 SAVE-SSD

Terminate subsystem definition(s)

Function

The SAVE-SSD statement terminates the sequence of statements used to define one or more subsystems initiated by the statements START-SSD-CREATION and SET-SUBSYSTEM-ATTRIBUTES.
This statement requests SSCM to save the subsystem definition(s) to the ISAM file named in the START-SSD-CREATION statement. This ISAM file is called the SSD object.

SAVE-SSD will be rejected if neither of the following statements has been executed beforehand:

- START-SSD-CREATION
- SET-SUBSYSTEM-ATTRIBUTES

SAVE-SSD is aborted and an error message is issued if the SSD object contains subsystems which

- have been defined with CLOSE-CTRL-ROUTINE=*DYNAMIC, but the holder task is not being used as a work task
- have different INSTALLATION-UNIT names (installed software units)

Format

```
SAVE-SSD
```

4.3.12 SEPARATE-ADDRESS-SPACE

Control disjunctive distribution of subsystems in class 5 memory

Function

This statement is used to control the disjunctive distribution of subsystems in class 5 memory. Using this statement, it is possible to prevent unwanted overlaps in the address spaces of subsystems which could be caused by the SET-SUBSYSTEM-ATTRIBUTES statements.

Use of this statement is irrelevant for subsystems in class 3 or class 4 memory (operand MEMORY-CLASS=*SYSTEM-GLOBAL in the SET-SUBSYSTEM-ATTRIBUTES statement), and for subsystems in class 6 memory (MEMORY-CLASS=*LOCAL-UNPRIVILEGED).
Subsystems in class 6 memory are never used in parallel, and may therefore overlap in the address space; subsystems in class 3 or 4 memory never overlap.

SET-SUBSYSTEM-ATTRIBUTES will be rejected if none of the following statements has been executed beforehand:

- START-SSD-CREATION
- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

Format

SEPARATE-ADDRESS-SPACE
SUBSYSTEM-NAME = <structured-name 1..8> ,**FROM-SUBSYSTEMS** = list-poss(15): <structured-name 1..8>

Operands

SUBSYSTEM-NAME = <structured-name 1..8>
Name of the subsystem which must not overlap with other subsystems.

The subsystem must be known to SSCM and must already be defined.

FROM-SUBSYSTEMS = list-poss(15): <structured-name 1..8>
List of a maximum of 15 subsystems, none of which may overlap with the subsystem named in the SUBSYSTEM-NAME operand.

Notes

The SEPARATE-ADDRESS-SPACE statement always creates at least two disjunctive relations, as can be seen from the following example:

```
//SEPARATE-ADDRESS-SPACE SUBSYSTEM-NAME=ONE,FROM-SUBSYSTEMS=(TWO,THREE)
```

Execution of this statement creates the following relations:

- the address space of subsystem ONE is non-overlapping with the address space of subsystem TWO
- the address space of subsystem ONE is non-overlapping with the address space of subsystem THREE
- the address space of subsystem TWO is non-overlapping with the address space of subsystem ONE

- the address space of subsystem THREE is non-overlapping with the address space of subsystem ONE

4.3.13 SET-SUBSYSTEM-ATTRIBUTES

Define attributes and entry points of subsystem

Function

This statement enables all the attributes and entry points for a subsystem to be set.

The way in which the definition of a new subsystem will be compiled depends on what statement preceded the definition:

- After a START-CATALOG-CREATION statement, the subsystem can be integrated into the **catalog**.
- After a START-SSD-CREATION statement, the subsystem can be integrated into the **SSD object**.
  However, it is not permissible to set up different versions of the same subsystem in the same SSD object.

Subsystems defined with the operand
MEMORY-CLASS=*SYSTEM-GLOBAL ..., SUBSYSTEM-ACCESS=*SYSTEM are privileged subsystems.

When setting up the definition, the following points should be noted:

- The name and version of the subsystem must not already be present in the subsystem catalog.
- Two different versions of one and the same subsystem cannot be defined in the same SSD object.
- The subsystem name CP is reserved for DSSM and must not be specified.
- The names of the entry points must not be present in duplicate.
- The name of the subsystem which is being defined must not be included in the list of subsystems to which there are address or dependency relations.
- Within this list, a subsystem name must not appear more than once.

SET-SUBSYSTEM-ATTRIBUTES is rejected if none of the following statements were executed beforehand:

- START-SSD-CREATION
- START-CATALOG-CREATION
- START-CATALOG-MODIFICATION

Note on syntax

The special data type <symbol>, which is described in detail in the “BLSSERV” manual, can also be used for the names of the entry points in the following operands (in the format, the data type is specified as <name>):

- LINK-ENTRY
- DEINIT-ROUTINE
- DYNAMIC-CHECK-ENTRY
- INTERFACE-VERSION
- INIT-ROUTINE
- SUBSYSTEM-ENTRIES
- CLOSE-CTRL-ROUTINE
- STOPCOM-ROUTINE

Format

SET-SUBSYSTEM-ATTRIBUTES

SUBSYSTEM-NAME = <structured-name 1..8>(...)

<structured-name 1..8>(...)

VERSION = <c-string 3..8> / <text 3..8>

,**INSTALLATION-UNIT** = ***NONE** / ***STD** / <text 1..30>

,**INSTALLATION-USERID** = ***NONE** / ***DEF**AULT**-USERID** / <name 1..8>

,**COPYRIGHT** = ***NONE** / <c-string 1..54>(...)

<c-string 1..54>(...)

YEAR = ***YEAR-1990** / <c-string 4..4>

,**LIB**RARY = ***STD** / ***CPLINK** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>

,**DEFAULT-NAME** = <filename 1..54 without-gen-vers>

,**SUBSYSTEM-LOAD-MODE** = ***ANY** / ***STD** / ***ADVANCED**

,**REP-FILE** = ***STD** / ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>

,**DEFAULT-NAME** = <filename 1..54 without-gen-vers> / ***NONE**

,**REP-FILE-MANDATORY** = ***NO** / ***Y**ES

,**MES**SAGE**-FILE** = ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>

,**DEFAULT-NAME** = <filename 1..54 without-gen-vers> / ***NONE**

,**SUBSYSTEM-INFO-FILE** = ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>

,**DEFAULT-NAME** = <filename 1..54 without-gen-vers> / ***NONE**

,**SYNTAX-F**ILE = ***NO** / ***INSTALLED**(...) / <filename 1..54 without-gen-vers>

***INSTALLED**(...)

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>

,**DEFAULT-NAME** = <filename 1..54 without-gen-vers> / ***NONE**

,**DYNAMIC-CHECK-ENTRY** = ***STD** / ***NO** / <text 1..8 without-sep>

,**CRE**ATION**-TIME** = ***AT-CRE** ATION **-REQ** UEST / ***AT-SUBSYS**TEM**-CALL**(...) / ***AT-DSSM-LOAD** /
***BEFORE-DSSM-LOAD** / ***MANDATORY-AT-STARTUP** / ***BEFORE-SYSTEM-READY** /
***AFTER-SYSTEM-READY**

***AT-SUBSYS**TEM**-CALL**(...)

ON-ACTION = ***STD** / ***ISL-CALL** / *ANY

,**INIT-ROUTINE** = ***NO** / <text 1..8 without-sep>

,**CLOSE-CTRL-ROUTINE** = ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**STOPCOM-ROUTINE** = ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**DEINIT-ROUTINE** = ***NO** / ***DYN**AMIC / <text 1..8 without-sep>

,**STOP-AT-SHUTDOWN** = ***NO** / ***Y**ES

,**INTERFACE-VERSION** = ***NO** / <text 1..8 without-sep>

,**SUBSYSTEM-HOLD** = ***ALLOW** ED / ***FORBIDDEN**

,**STATE-CHANGE-CMDS** = ***ALLOW** ED / ***FORBIDDEN** / ***BY-ADMINISTRATOR-ONLY**

,**FORCED-STATE-CHANGE** = ***FORBIDDEN** / ***ALLOW**ED

,**RESET** = ***FORBIDDEN** / ***ALLOW**ED

,**RESTART-REQUIRED** = ***NO** / ***Y**ES

,**VERSION-COEXISTENCE** = ***FORBIDDEN** / ***ALLOW**ED

,**VERSION-EXCHANGE** = ***FORBIDDEN** / ***ALLOW**ED

,**SUBSYSTEM-ENTRIES** = ***NONE** / list-poss(100): <text 1..8>(...) **/ *BY-PROGRAM(...)**

<text 1..8>(...)

MODE = ***LINK** / ***ISL**(...) / ***SVC**(...) / ***SYSTEM-EXIT**(...)

***ISL**(...)

FUNCTION-NUMBER = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SVC**(...)

NUMBER = <integer 0..255>

,**CALL-BY-SYSTEM-EXIT** = ***ALLOW** ED / ***FORBIDDEN**

,**FUNCTION-NUMBER** = ***NONE** / <integer 0..255>(...)

<integer 0..255>(...)

FUNCTION-VERSION = <integer 1..255>

***SYSTEM-EXIT**(...)

NUMBER = <integer 0..127>

,**CONNECTION-ACCESS** = ***ALL** / ***SYS**TEM / ***SIH**

,**CONNECTION-SCOPE** = ***TASK** / ***PROG**RAM / ***FREE** / ***CALL** / ***OPTIMAL**

, **FIRST-CONNECTION** = ***ALLOW** ED / ***FORBIDDEN**

***BY-PROGRAM(...)**

CONNECTION-SCOPE = ***TASK** / ***PROGRAM**

,**MEMORY-CLASS** = ***SYSTEM-GLOBAL** (...) / ***LOCAL-PRIVILEGED**(...) / ***LOCAL-UNPRIVILEGED**(...) /

***BY-SLICE**(...)

***SYSTEM-GLOBAL**(...)

SUBSYSTEM-ACCESS = ***LOW** / ***SYS**TEM / ***HIGH**

***LOCAL-PRIVILEGED**(...)

SIZE = <integer 1..32767>

***LOCAL-UNPRIVILEGED**(...)

SIZE = <integer 1..32767>

,**SUBSYSTEM-ACCESS** = ***LOW** / ***HIGH**

,**START-ADDRESS** = ***ANY** / <x-string 7..8>

***BY- SLICE**(...)

SIZE = <integer 1..32767>

,**LINK-ENTRY** = <text 1..8 without-sep>(...)

<text 1..8 without-sep>(...)

AUTOLINK = ***FORBIDDEN** / ***ALLOW**ED

,**REFERENCED-SUBSYSTEM** = ***NONE** / list-poss(15): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***LOW** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***HIGH** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**UNRESOLVED-EXTERNALS** = ***FORBIDDEN** / ***ALLOW**ED

,**CHECK-REFERENCE** = ***Y** ES / ***NO**

,**RELATED-SUBSYSTEM** = ***NONE** / list-poss(100): <structured-name 1..8>(...)

<structured-name 1..8>(...)

LOWEST-VERSION = ***LOW** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

,**HIGHEST-VERSION** = ***HIGH** EST **-EXIST** ING / <c-string 3..8> / <text 3..8>

Operands

SUBSYSTEM-NAME = <structured-name 1..8>(...)
Specifies the name and version of the subsystem which is to be defined.

VERSION = <c-string 3..8> / <text 3..8>
The version of the subsystem must be specified in the format “[V][n]n.m[ann]”. The text elements have the following meanings:

nn = Main version (numeric)
m = Revision version (numeric)
ann = Update status (a=letter, release status; nn=numeric, correction status)

INSTALLATION-UNIT =
Defines the name of the installed software unit. A value other than *NONE must be specified for all subsystems installed with IMON, and likewise if the value
*INSTALLED(LOGICAL-ID=...) was defined for the operand SUBSYSTEM-LIBRARY, REP-FILE, SUBSYSTEM-INFO-FILE, MESSAGE-FILE or SYNTAX-FILE.
The syntax rules described in the “IMON“ manual must be observed when defining the name.

INSTALLATION-UNIT = *NONE
Default setting: no name is assigned. This default setting is not allowed for any subsystems installed with IMON.

INSTALLATION-UNIT = *STD
The name specified via the SUBSYSTEM-NAME operand is used as the name of the installed software unit.

INSTALLATION-UNIT = <text 1..30>
Name of the installed software unit.

INSTALLATION-USERID =
Defines a user ID under which the relevant DSSM task expects the subsystem satellites (REP file, object module library, message file, syntax file and subsystem information file) if these files have not yet been assigned to a user ID, i.e. the file name was specified without a user ID.

INSTALLATION-USERID = *NONE
Default value: the files will not be expected under a specific user ID.

INSTALLATION-USERID = *DEFAULT-USERID
The files are expected under the default system ID (prefix “$.”) or under the user ID of the calling task if the subsystem is a local subsystem.

INSTALLATION-USERID = <name 1..8>
User ID under which the subsystem satellites are to be expected.
If this statement applies to an SSD object, the files will only actually be expected under the user ID specified here if no user ID was specified in the ADD-CATALOG-ENTRY statement (inclusion of subsystem definitions from the SSD object in the catalog). The ID specified in ADD-CATALOG-ENTRY takes precedence.

COPYRIGHT =
Specifies whether or not a copyright notice is to be displayed when the subsystem is started and, if so, which one.

COPYRIGHT = *NONE
Default value: no copyright notice is to be output.

COPYRIGHT = <c-string 1..54>(...)
Text of the copyright notice which is to be output together with the creation date when the subsystem is started.

YEAR = *YEAR-1990 / <c-string 4..4>
Number of the year which is to appear in the statement as the creation date. This is not subjected to a semantic check.

LIBRARY =
Specifies the name of the program or object module library (OML) from which the object code for the subsystem is to be loaded when it is activated.

LIBRARY = *STD
Default value: when the subsystem is started, the object code will automatically be loaded from the library SYSLNK.<subsysname>.<subsysvers#>. This library is stored under the user ID under which the holder task is running. For local subsystems this is the user ID of the caller, and for global subsystems it is TSOS. “<subsysvers#>” is a three-character value consisting of the elements “mmm” (for the operand SUBSYSTEM-NAME=...(VERSION=...)).

LIBRARY = *CPLINK
The subsystem which is to be defined is linked to the BS2000 control program (CP) and must have been loaded before DSSM was activated. This operand may only be used in conjunction with the operand CREATION-TIME=*BEFORE-DSSM-LOAD.

LIBRARY = *INSTALLED(...)
The library name must be determined by calling IMON (administration of installation paths). If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to the subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the program library or object module library.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
New library name if IMON-GPN is not available or if the logical name is unknown.

LIBRARY = <filename 1..54 without-gen-vers>
Fully qualified file name of the object module library from which the object code for the subsystem is to be loaded.

SUBSYSTEM-LOAD-MODE =
Specifies the load mode of the subsystem (via the BLS-DSSM interface $PBBND1).

SUBSYSTEM-LOAD-MODE = *ANY
Default value: the BLS (Binder-Loader-Starter system) is called up in STD run mode and loads the subsystem as an object module. If an error occurs during loading, BLS will be called a second time. The call takes place in ADVANCED run mode, and the subsystem is loaded as a link and load module (LLM).
Support for this operand value is provided only for the sake of compatibility.

SUBSYSTEM-LOAD-MODE = *STD
The BLS is called up in STD run mode and loads the subsystem as an object module.

SUBSYSTEM-LOAD-MODE = *ADVANCED
The BLS is called up in ADVANCED run mode and loads the subsystem as a link and load module (LLM).

REP-FILE =
Specifies whether or not system REPs are required for the subsystem which is being defined, and identifies the file in which they are stored. These correction statements are used during activation of the subsystem, and are applied solely to the modules stored and loaded in the object module library, not to other subsystems or the BS2000 control program. A REP file can also be specified for modules of a nonprivileged subsystem.
REP-FILE must not be specified together with LIBRARY=*CPLINK.

REP-FILE = *STD
Unless another file is specified, system REPs are loaded from the REP file with the nameSYSREP.<subsysname>.<subsysvers#>. This REP file is stored under the user ID under which the holder task is running. For local subsystems this is the user ID of the caller, and for global subsystems it is TSOS.
“<subsysvers#>” is a three-character value consisting of the elements “mmm” specified for the operand SUBSYSTEM-NAME=...(VERSION=...).

REP-FILE = *NO
No REP files are to be processed for the subsystem.

REP-FILE = *INSTALLED(...)
The name of the REP file must be determined by calling IMON-GPN (administration of installation paths). If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the REP file.

DEFAULT-NAME =
Name of the REP file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE
No new name is assigned.

REP-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the REP file from which the correction statements are to be read.

REP-FILE-MANDATORY =
Specifies whether or not a REP file declared via the REP-FILE operand is to be processed when loading the subsystem.

REP-FILE-MANDATORY = *NO
Default value: the use of a REP file is not mandatory, i.e. neither the REP file nor its entries are to be checked when the subsystem is activated. Even if the REP file can‘t be accessed or if individual correction statements are invalid, the subsystem will still be started.

REP-FILE-MANDATORY = *YES
If any of the following errors occurs during processing of the REP file, any attempt to load the subsystem will be terminated:

- the REP file is not cataloged, or it cannot be read
- checking of the correction statements reveals an error
- the name of a correction statement is invalid

- DMS reports an error during access to the NOREF file (this file is used during loading of a subsystem to prevent invalid system REPs from being logged at the operator terminal)

MESSAGE-FILE =
Specifies whether there is a subsystem-specific message file which is to be automatically activated when the subsystem is loaded. For subsystems which are defined with the creation time AT-DSSM-LOAD, a dependency relation to the MIP subsystem must be specified in the RELATED-SUBSYSTEMS operand.

MESSAGE-FILE = *NO
Default value: no message file is to be activated. This value is mandatory for all subsystems which are defined with the creation time BEFORE-DSSM-LOAD (see also the CREATION-TIME operand), as it is not possible to activate a message file as early as this.

MESSAGE-FILE = *INSTALLED(...)
The name of the message file must be determined by calling IMON-GPN (administration of installation paths).
If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to the subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the message file.

DEFAULT-NAME =
Name of the message file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE
No new name is assigned.

MESSAGE-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the message file. This will be automatically activated when the subsystem is loaded (START-SUBSYSTEM command) and automatically deactivated when it is unloaded (STOP-SUBSYSTEM).

SUBSYSTEM-INFO-FILE =
Specifies whether or not a subsystem information file (SSINFO) is available. This file contains subsystem-specific data (subsystem satellites and configuration data) which cannot be processed centrally by DSSM.

SUBSYSTEM-INFO-FILE = *NO
Default value: no information file is available for the subsystem.

SUBSYSTEM-INFO-FILE = *INSTALLED(...)
The name of the information file must be determined by calling IMON-GPN (administration of installation paths).
If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the information file.

DEFAULT-NAME =
Name of the information file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE
A new name is assigned.

SUBSYSTEM-INFO-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the information file. This name is automatically passed to the activation and deactivation routines (INIT-/DEINIT-/STOPCOM-/CLOSE-CTRL-ROUTINE operands) when they are called.

SYNTAX-FILE =
Specifies whether the subsystem has linked to it a syntax file which will be activated automatically when the subsystem is loaded. For subsystems which are defined with the start attribute MANDATORY-AT-STARTUP, a dependency relation to the SDF subsystem must be declared in the RELATED-SUBSYSTEMS operand.

SYNTAX-FILE = *NO
Default value: no syntax file is to be activated. This value is mandatory for all subsystems which are defined with the start time BEFORE-DSSM-LOAD or AT-DSSM-LOAD (see also the CREATION-TIME operand), as it is not possible to activate a syntax file as early as this.

SYNTAX-FILE = *INSTALLED(...)
The name of the syntax file must be determined by calling IMON-GPN (administration of installation paths).
If one of the subsystem satellites is referenced with a logical ID, logical IDs must be specified for all satellites belonging to this subsystem. If a logical ID is assigned, a value other than *NONE must be assigned for the INSTALLATION-UNIT operand.

LOGICAL-ID = <filename 1..30 without-catid-userid-gen-vers>
Logical ID of the syntax file.

DEFAULT-NAME =
Name of the syntax file if IMON-GPN is not available or if the logical ID is unknown.

DEFAULT-NAME = <filename 1..54 without-gen-vers>
A new name is assigned.

DEFAULT-NAME = *NONE
No new name is assigned.

SYNTAX-FILE = <filename 1..54 without-gen-vers>
Fully qualified name of the syntax file which is to be automatically activated when the subsystem is loaded.

DYNAMIC-CHECK-ENTRY =
Specifies whether a dynamic identity check is to be carried out on the subsystem. For this purpose, an entry point must be specified, at which both the subsystem name (eight characters) and also the version number (four or seven characters) must be located. DSSM checks whether the identification specified in the definition agrees with the subsystem which is loaded.

DYNAMIC-CHECK-ENTRY = *STD
By default, the entry point specified in the LINK-ENTRY operand will be applied in carrying out the identity check.

DYNAMIC-CHECK-ENTRY = *NO
No check is to be carried out. However, this value of the operand must not be used for any subsystems which are loaded before DSSM is activated (CREATION-TIME=*BEFORE-DSSM-LOAD).

DYNAMIC-CHECK-ENTRY = <text 1..8 without-sep>
Name of the entry point which is to be used in applying the identity check.

CREATION-TIME =
Specifies the point in time at which activation of the subsystem (CREATE routine) is initiated.
There are two separate phases during system initialization when DSSM takes over the control of system initialization after it has been called by the startup routine:

Phase 1: The DSSM code is loaded, the DSSM task is generated and started.
This task reserves class 5 memory, reads in the subsystem catalog, and starts those subsystems which have been defined with the start attributes BEFORE-DSSM-LOAD and AT-DSSM-LOAD.
After these subsystems have been loaded, control of system initialization returns to the startup routine.

Phase 2: Following a second call, all those subsystems are loaded which have been defined with the start attributes MANDATORY-AT-STARTUP, BEFORE-SYSTEM-READY and AFTER-SYSTEM-READY.
In the case of the first two of these start attributes, loading of the subsystems is synchronized with the startup routine (i.e. loading must be completed), but for the last of them asynchronous loading is initiated.
Control of system initialization returns to the startup routine.

If different versions of a subsystem are to be defined, it is only possible to specify the start attributes, for use in phases 1 and 2 of system initialization, for one of these versions.

CREATION-TIME = *AT-CREATION-REQUEST
Default value: the subsystem must be explicitly loaded by means of the START-SUBSYSTEM command.

CREATION-TIME = *AT-SUBSYSTEM-CALL(...)
The subsystem is to be automatically loaded when the first SVC or ISL call is made. This operand value is reserved for subsystems which are called via SVC or ISL.

If two or more versions of a subsystem are defined with this operand value,
VERSION-COEXISTENCE=*ALLOWED must be specified for all of these versions and FUNCTION-NUMBER and FUNCTION-VERSION must be specified for their SVC or ISL entry points, which were declared with CONNECTION-ACCESS with a value other than *SIH.
At least one of the specified subsystems must have been declared with
SUBSYSTEM-ENTRIES ..., MODE=*SVC/*ISL (corresponding to the value of the ON-ACTION operand).

ON-ACTION =
Determines what initiates automatic loading of the subsystem.

ON-ACTION = *STD
Default setting: loading begins when any SVC entry point belonging to the subsystem is called.

ON-ACTION = *ISL-CALL
Loading begins when any ISL entry point belonging to the subsystem is called.

ON-ACTION = *ANY
Loading begins when any SVC or ISL entry point belonging to the subsystem is called.

CREATION-TIME = *AT-DSSM-LOAD
The subsystem is to be loaded during system initialization (phase 1) under the control of the DSSM task.
The subsystem must be a privileged one, and may only have address or dependency relations to subsystems which are also defined with this start attribute or with the start attribute BEFORE-DSSM-LOAD.
The files for this subsystem must be held on the home pubset under the TSOS user ID, because at the time of startup the user catalog is not accessible and IMPORT-PUBSET processing has not been completed.
Specification of a syntax file is not permitted for these subsystems.

CREATION-TIME = *BEFORE-DSSM-LOAD
The subsystem is to be loaded during system initialization (phase 1), but not under the control of the DSSM task. Such subsystems are linked to the control program, and do not need to be synchronized with the DSSM task when activated. However, after the subsystem is loaded it again runs under the control of DSSM and can, from the user's point of view, be controlled in the same way as other subsystems.
No address or dependency relations are possible to subsystems which are defined with any other start attribute. Nor is it permissible to link in a message or syntax file. All job entries (SUBSYSTEM-ENTRIES operand) must be declared, because DSSM creates the (privileged) connection to these job entries. Responsibility for ensuring that at least one version of this subsystem is available at any given time (e.g. PLAM) rests entirely with the subsystem developer.

The name of the link context for these subsystems must be unique because DSSM must also honor an unload request even if DSSM has not loaded the subsystem code. An entry point (operand DYNAMIC-CHECK-ENTRY) must have been specified.

CREATION-TIME = *BEFORE-SYSTEM-READY
The subsystem is to be loaded during system initialization (phase 2). Activation is initiated synchronously; not until loading is complete (or a load error occurs) does control return to the startup routine, which can then report "SYSTEM READY".
The subsystem must be privileged, and may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD, AT-DSSM-LOAD or MANDATORY-AT-STARTUP. The files for this subsystem must be cataloged on the home pubset.
If a nonprivileged subsystem is declared with this operand value, it is automatically given the value *AFTER-SYSTEM-READY. SSCM issues a message.

CREATION-TIME = *MANDATORY-AT-STARTUP
The subsystem must be loaded during system initialization (phase 2). Activation is initiated synchronously - as in the case of BEFORE-SYSTEM-READY. By contrast with the latter however, loading of the subsystem must in this case be completed **successfully**. Otherwise a message is passed to the startup routine reporting that a mandatory subsystem could not be loaded. In this case, the startup routine will decide whether processing should continue or be terminated.
The subsystem must be privileged, and may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD or AT-DSSM-LOAD. The files for this subsystem must be cataloged on the home pubset.

CREATION-TIME = *AFTER-SYSTEM-READY
The loading of this subsystem is to be initiated during system initialization (phase 2). Execution of this routine is not synchronized with the startup routine, which can report "SYSTEM READY" before subsystem loading is complete.
The subsystem may only have address or dependency relations to subsystems defined with the same attribute or with the start attribute BEFORE-DSSM-LOAD, AT-DSSM-LOAD, MANDATORY-AT-STARTUP or BEFORE-SYSTEM-READY. The files for this subsystem must be cataloged on the home pubset.

INIT-ROUTINE =
Specifies whether there is an initialization routine for the subsystem which must be performed when it is started or resumed. In this case, the name of an entry point must be known, and DSSM will delegate initialization to the holder task of the subsystem concerned. It is strongly recommended that an entry point be defined for all subsystems which have the start attribute BEFORE-DSSM-LOAD. During loading of the subsystem (i.e. when the initialization routine is carried out), the subsystem is then informed that DSSM can assume control over the opening and closing of relations.

An INIT routine must be declared with RESTART-REQUIRED=*YES.

INIT-ROUTINE = *NO
Default value: no initialization routine is run.

INIT-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the initialization routine.
In the holder task, control is passed to the initialization routine, so that the subsystem can initialize itself. To permit this, it is passed:

- the name and version of the subsystem, as defined in SSCMCAT
- the name of the SSINFO file, if one was specified in the SUBSYSTEM-INFO-FILE operand
- the address of the entry point specified during loading and linking (LINK-ENTRY operand)
- the link context name used by the dynamic binder loader
- the name of the memory pool (for subsystems in class 5 or class 6 memory), in order that the subsystem can refer to its own selectable units/load units during dynamic loading
- the name of the message file
- the address of the SUBSYSTEM-PARAMETER operand, if a string has been specified in the START-SUBSYSTEM command

At the end of initialization, a return message is expected from the subsystem, indicating whether initialization was successful and whether the holder task is to be used as a work task (as specified in the ASSIGN-HOLDER-TASK statement). Depending on what is reported here, the task will from then on be controlled by DSSM or by the subsystem.

CLOSE-CTRL-ROUTINE =
Specifies whether a special routine is to be run for the subsystem if it is suspended or deactivated.
If a subsystem is deactivated (by a STOP-SUBSYSTEM or HOLD-SUBSYSTEM command), DSSM passes control to this routine at the identified entry point in the holder task, or (for *DYNAMIC) this is reported via a bourse or FITC link (as determined by return messages during initialization).

The parameters passed are the same ones as are passed for the INIT-ROUTINE operand. Branching to this routine ensures that a connection to the subsystem still exists.
If a CLOSE-CTRL routine exists, it is possible to change versions without interrupting the BS2000 session. At any given point in time, there is exactly one valid version (either the old version is still available, or the new version has already become available). Without a CLOSE-CTRL routine, changing versions always entails interrupting the connection so that the STOPCOM routine of the old version and the INIT routine of the new version can execute (see also "Swapping subsystem versions").

CLOSE-CTRL-ROUTINE = *NO
Default value: the subsystem designates no routine which is to be run when the subsystem is deactivated or suspended.

CLOSE-CTRL-ROUTINE = *DYNAMIC
This routine is called up via the bourse or FITC. The subsystem passes the required parameters to the CLOSE-CTRL routine dynamically at the end of the INIT routine, and DSSM is informed of the identity of the bourse or FITC port.
In order for the CLOSE-CTRL routine to run, an INIT routine (INIT-ROUTINE operand) and a STOPCOM routine (operand STOPCOM-ROUTINE=*NO/*DYNAMIC) must have been specified.
The holder task of the subsystem must be a work task when the CLOSE-CTRL routine is called (ASSIGN-HOLDER-TASK statement).

CLOSE-CTRL-ROUTINE = <text 1..8 without-sep>
Name of the entry point of the relevant subsystem routine.

STOPCOM-ROUTINE =
Specifies whether the subsystem incorporates a routine which can carry out active termination of tasks.
If a subsystem is deactivated (by a STOP-SUBSYSTEM or HOLD-SUBSYSTEM), then DSSM passes control to this routine at the identified entry point in the holder task, or (for *DYNAMIC) this is reported via a bourse or FITC link (as determined by return messages during initialization).
The parameters which are passed are the same as for the INIT-ROUTINE operand. Branching to this routine ensures that calling tasks will no longer be connected to the subsystem. Tasks which still have outstanding call relations to the subsystem are unaffected by this.
If CLOSE-CTRL-ROUTINE=*DYNAMIC is declared, the operand STOPCOM-ROUTINE=*NO/*DYNAMIC must be specified.

STOPCOM-ROUTINE = *NO
Default value: the subsystem concerned incorporates no such routine.

STOPCOM-ROUTINE = *DYNAMIC
This routine is called via the bourse or FITC. The subsystem passes the required parameters to the STOPCOM routine dynamically at the end of the CLOSE-CTRL routine or (if none exists) at the end of the INIT routine. DSSM is informed of the identity of the bourse or FITC port.

A prerequisite for the use of the STOPCOM routine is than an INIT routine has been specified (INIT-ROUTINE operand). When the STOPCOM routine is called, the holder task for the subsystem must be used as a work task (ASSIGN-HOLDER-TASK statement).

STOPCOM-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the subsystem routine concerned.

DEINIT-ROUTINE =
Specifies whether the subsystem incorporates a routine which can carry out deinitialization of the subsystem. This deinitialization routine causes the resources which were requested by the subsystem (memory, files, devices) to be returned.
If a subsystem is deactivated (by a STOP-SUBSYSTEM or HOLD-SUBSYSTEM command), then DSSM passes control to this routine at the identified entry point in the holder task, or (for *DYNAMIC) this is reported via a bourse or FITC link (as determined by return messages during initialization).
If a subsystem is defined with an INIT routine and a CLOSE-CTRL routine, a DEINIT routine - with the same operand value as the CLOSE-CRTL routine - must be specified.
The parameters which are passed are the same as for the INIT-ROUTINE operand. Branching to this routine ensures that calling tasks will no longer be connected to the subsystem and all existing call relations to the subsystem are deleted.

DEINIT-ROUTINE = *NO
Default value: the subsystem concerned does not incorporate a deinitialization routine to request the release of resources.

DEINIT-ROUTINE = *DYNAMIC
The routine is called via the bourse or FITC.The subsystem passes the required parameters to the DEINIT routine dynamically at the end of the STOPCOM routine or, if none exists, at the end of the CLOSE-CTRL routine or, if neither a STOPCOM nor a CLOSE-CTRL routine is incorporated, at the end of the INIT routine. DSSM is informed of the identity of the bourse or FITC port.
A prerequisite for the use of the DEINIT routine is than an INIT routine has been specified (INIT-ROUTINE operand). When the DEINIT routine is called, the holder task for the subsystem must be used as a work task (ASSIGN-HOLDER-TASK statement).

DEINIT-ROUTINE = <text 1..8 without-sep>
Name of the entry point to the subsystem routine concerned.

STOP-AT-SHUTDOWN =
Specifies whether the subsystem is to be unloaded automatically at shutdown after the user tasks have terminated.

STOP-AT-SHUTDOWN = *NO
Default value: the subsystem will not be unloaded automatically.
This parameter should not be specified for subsystems which have address relations to other subsystems which are defined with STOP-AT-SHUTDOWN=*YES.

STOP-AT-SHUTDOWN = *YES
The subsystem will be unloaded automatically at shutdown.
This specification will be ignored if no STOPCOM, DEINIT or CLOSE-CRTL routine is specified.

INTERFACE-VERSION =
Identifies the entry point via which DSSM can access the interface version which is to be used for calling the INIT, DEINIT, STOPCOM or CLOSE-CTRL routine.
This operand is mandatory for subsystems for which an entry point was specified (INIT, CLOSE-CTRL, STOPCOM, DEINIT routine).

INTERFACE-VERSION = *NO
Default value: the subsystem does not call a INIT, DEINIT, STOPCOM or CLOSE-CTRL routine.

INTERFACE-VERSION = <text 1..8 without-sep>
Name of the entry point. The entry point points to the standard header where the interface version is stored. The standard header is generated by calling the macro $ESMINT(I) with MF=I/L. This operand is mandatory for subsystems for which an INIT, DEINIT, STOPCOM or CLOSE-CTRL routine was defined.

SUBSYSTEM-HOLD =
Specifies whether the subsystem which is loaded may be suspended or unloaded.

SUBSYSTEM-HOLD = *ALLOWED
Default value: the subsystem which is loaded may be suspended and unloaded. The commands HOLD-SUBSYSTEM and STOP-SUBSYSTEM are permissible for this subsystem.

SUBSYSTEM-HOLD = *FORBIDDEN
The commands HOLD-SUBSYSTEM and STOP-SUBSYSTEM must not be used for this subsystem; it will only be unloaded at shutdown - as specified by the STOP-AT-SHUTDOWN operand.

Unloading the subsystem by replacing it with another subsystem entails no interruption.

STATE-CHANGE-CMDS =
Specifies whether or not the DSSM commands for controlling the subsystem (START-SUBSYSTEM, STOP-SUBSYSTEM, HOLD-SUBSYSTEM and RESUME-SUBSYSTEM) may be used during a session.
If a change is made from one version of a subsystem to another, the value specified for STATE-CHANGE-CMDS for the version being replaced is ignored.

STATE-CHANGE-CMDS = *ALLOWED
Default value: the commands may be used from the operator terminal and under the privileged user ID (the user ID which has the SUBSYSTEM-MANAGEMENT system privilege).

STATE-CHANGE-CMDS = *FORBIDDEN
The commands must not be used - neither from the operator terminal nor under the privileged user ID.

STATE-CHANGE-CMDS = *BY-ADMINISTRATOR-ONLY
The commands may only be used under the privileged user ID; the commands are not available to the operator at the operator terminal.

FORCED-STATE-CHANGE =
Specifies whether use of the operand FORCED=*YES is permitted within the commands STOP-SUBSYSTEM and HOLD-SUBSYSTEM. This function can be used to force the unconditional deactivation of the subsystem.

FORCED-STATE-CHANGE = *FORBIDDEN
Default value: it is not possible to force deactivation of the subsystem. DSSM will reject any use of the FORCED operand in the commands concerned, and will issue a corresponding error message.

FORCED-STATE-CHANGE = *ALLOWED
The operand FORCED=*YES may be used for this subsystem.
This operand value must not be used in conjunction with SUBSYSTEM-HOLD=*FORBIDDEN.

RESET =
Specifies whether the operand RESET=*YES is permitted within the commands START-SUBSYSTEM and RESUME-SUBSYSTEM. This function can be used to force the unconditional loading or resumption of the subsystem, even if the state of the subsystem is currently IN-DELETE or IN-HOLD.

RESET = *FORBIDDEN
Default value: it is not possible to force the activation of the subsystem. DSSM will reject the use of the RESET operand in the commands concerned, and will issue a corresponding error message.

RESET = *ALLOWED
The operand RESET=*YES may be used for this subsystem.
This operand value must not be specified together with SUBSYSTEM-HOLD=*FORBIDDEN.

RESTART-REQUIRED =
Specifies whether the initialization routine for the subsystem is to be executed if the holder task terminates abnormally.

RESTART-REQUIRED = *NO
Default value: the initialization routine is not used to restart the subsystem.

RESTART-REQUIRED = *YES
If the holder task terminates abnormally, the initialization routine should be used. Provision must have been made in the INIT-ROUTINE operand for executing this routine.

VERSION-COEXISTENCE =
Specifies whether more than one version of the same subsystem may be active simultaneously.

VERSION-COEXISTENCE = *FORBIDDEN
Default value: the current version of the subsystem cannot coexist with another version of the same subsystem.

VERSION-COEXISTENCE = *ALLOWED
The current version of the subsystem can coexist with another version of the same subsystem (coexistence mode). In the definition of the job entry point (SUBSYSTEM-ENTRIES operand), indirect links via system exit routines must not have been specified. If different versions of the same subsystem are loaded and the same job entry point is defined for these, the link which is implemented is always to the highest loaded version of the subsystem.
If coexistent subsystems access coexistent syntax files, the latter must have been declared in the SSD object and cannot be administered by SDF.
However, where the links are via SVC and ISL, it is possible to select a version using the operands FUNCTION-NUMBER and FUNCTION-VERSION.

VERSION-EXCHANGE =
Specifies whether a subsystem may be loaded in exchange mode. Exchange mode allows the temporary coexistence of two versions of the same subsystem. If version B of a subsystem is loaded whilst version A of the subsystem is already active, all new callers will be connected to version B. Jobs which are connected to version A will still be processed. When all the jobs which use version A have been processed, this will automatically be terminated.

In the definition it should be noted that the “old” version which is being replaced must not be dependent on the “new” version which replaces it.

VERSION-EXCHANGE = *FORBIDDEN
Default value: the current version of the subsystem must not be replaced.

VERSION-EXCHANGE = *ALLOWED
Exchange mode, which allows the temporary coexistence of two subsystems, is permitted for the current subsystem version.

SUBSYSTEM-ENTRIES =
Specifies the entry points (job entries) which are linked to the subsystem. Up to 100 entry points can be declared. If more than 100 entry points are desired for a subsystem, the additional ones can be defined by means of the statement MODIFY-SUBSYSTEM-ATTRIBUTES (for a subsystem in the catalog) or the statement ADD-SUBSYSTEM-ENTRIES (for a subsystem in an SSD object).

SUBSYSTEM-ENTRIES = *NONE
Default value: no new job entries are to be declared.

SUBSYSTEM-ENTRIES = list-poss(100): <text 1..8>
Declares the names of the entry points for a maximum of 100 job entries; the type of each of these must be defined in the substructures.

MODE =
Defines the type of a job entry point which is defined for the subsystem.

MODE = *LINK
Default value: the job entry point cannot be accessed by indirect linkage, but only by using a CONNECT relation through an external linkage editor symbol.
In the case of different versions of the same subsystem which use the same external linkage editor symbol, DSSM automatically sets up the link to the highest loaded version of the subsystem.

MODE = *ISL(...)
The job entry is effected by indirect linkage via System Procedure Linkage (for privileged subsystems only). If the specification includes in addition a function and version number for the ISL entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem
(if VERSION-COEXISTENCE=*ALLOWED is specified).

For different subsystems, if the job entry point is to be accessed by the same ISL entry point, they must be uniquely identified by specifying the function and version numbers.In the case of different versions of the same subsystem which use the same ISL entry point, then - if the function and version numbers are not specified - DSSM will automatically set up a connection to the highest loaded version of the subsystem.

In the case of different versions of the same subsystem which use the same ISL entry point and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller’s parameter list.

The value *ALL for the CONNECTION-ACCESS operand is not permissible for ISL entry points.

FUNCTION-NUMBER =
Specifies whether a particular function and version number of the ISL entry point is to be addressed, as the same ISL entry point can be used by different functions.

FUNCTION-NUMBER = *NONE
Default value: no particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)
Number of the ISL entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>
Version of the specified ISL function number.

MODE = *SVC(...)
Job entry is to be effected by an indirect connection using a supervisor call (SVC).
If the specification includes in addition a function and version number for the SVC entry point, the combination of entry point name, function and version numbers must not match any other combination for the various other subsystems in the catalog or the various versions of the same subsystem
(if VERSION-COEXISTENCE=*ALLOWED is specified).
For different subsystems, if the job entry point is to be accessed by the same SVC, they must be uniquely identified by specifying the function and version numbers.
In the case of different versions of the same subsystem which use the same SVC, then - if the function and version numbers are not specified - DSSM will automatically set up a connection to the highest loaded version of the subsystem.
In the case of different versions of the same subsystem which use the same SVC and for which the function and version numbers are not equal to *NONE, the version to which the connection is set up will be selected in accordance with the function and version numbers stored in the standard header of the caller’s parameter list.
If this operand value is specified, it is better to set the operand CONNECTION-ACCESS to the value *SYSTEM, instead of to *ALL.

NUMBER = <integer 0..255>
Number of the SVC via which job entry is to be effected. No SVC number greater than 191 may be used in conjunction with CONNECTION-ACCESS=*ALL.

CALL-BY-SYSTEM-EXIT =
Defines whether the specified SVC number may be called from within system exit routines.

CALL-BY-SYSTEM-EXIT = *ALLOWED
Default value: system exit routines are permitted to call the specified SVC number.

CALL-BY-SYSTEM-EXIT = *FORBIDDEN
System exit routines are not permitted to call the specified SVC number.

FUNCTION-NUMBER =
Specifies whether a particular function and version number of the SVC entry point is to be addressed, as the same SVC entry point can be used by different functions.

FUNCTION-NUMBER = *NONE
Default value: no particular function or version number is to be addressed.

FUNCTION-NUMBER = <integer 0..255>(...)
Number of an SVC entry point. The version must be nominated in the substructure which follows.

FUNCTION-VERSION = <integer 1..255>
Version of the specified SVC function number.

MODE = SYSTEM-EXIT(...)
Job entry is to be effected by an indirect connection using system exit routines.
This operand must not be used in conjunction with CONNECTION-ACCESS=*ALL.

NUMBER = <integer 0..127>
Number of the system exit routine.

CONNECTION-ACCESS =
Specifies the access authorization (privileges) required by the subsystem.

CONNECTION-ACCESS = *ALL
Default value: privileged and nonprivileged program runs may access the subsystem.This operand value must not be used in conjunction with MODE=*SYSTEM-EXIT/*ISL/*SVC (with an SVC number greater than 191).

CONNECTION-ACCESS = *SYSTEM
Only privileged program runs may access the subsystem.

CONNECTION-ACCESS = *SIH
Only tasks running in the SIH processor state may access the subsystem.
The subsystem called also runs in the SIH processor state, i.e. it is uninterruptible.

This operand value is permissible only for subsystems for which the entry point is defined via:

- System Procedure Linkage (MODE=*ISL(FUNCTION-NUMBER=*NONE))
- CONNECTION-SCOPE=*OPTIMAL
- MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=*SYSTEM)

CONNECTION-SCOPE =
Identifies the event which will call up the automatic cleardown of the connection to the specified subsystem job entry.

CONNECTION-SCOPE = *TASK
Default value: the connection will be cleared when the task terminates.

CONNECTION-SCOPE = *PROGRAM
The connection will be cleared when the program terminates, or before.
Only CONNECTION-SCOPE=*PROGRAM may be specified in conjunction with MEMORY-CLASS=*LOCAL-UNPRIVILEGED.
This operand value is recommended for subsystems which were declared with SUBSYSTEM-ACCESS=*LOW/*HIGH or MEMORY-CLASS=*BY-SLICE.

CONNECTION-SCOPE = *FREE
DSSM is not to carry out any checking of the connections to the job entry point. The connection will not be automatically cleared - unless explicitly requested. To avoid problems or possible errors when the subsystem is being unloaded, the connections must be managed by the subsystem itself.

CONNECTION-SCOPE = *CALL
On return from this job entry point, DSSM will automatically clear the connections.
This operand value is only available with subsystems for which the job entry is defined by means of System Procedure Linkage (ISL) or supervisor calls (SVC).

CONNECTION-SCOPE = *OPTIMAL
The subsystem is deactivated or suspended when there are no further tasks with a connection to this entry point.
A routine with an entry point defined with *OPTIMAL must be terminated with RETURN.If an entry point of a subsystem is defined with CONNECTION-SCOPE=*OPTIMAL, all of its entry points should be defined in the subsystem catalog with MODE !=*LINK. While a subsystem is deactivated or suspended, no call of the subsystem with CONNECTION-SCOPE=*OPTIMAL is accepted.

FIRST-CONNECTION =
Determines whether or not first connection of the task to the specified job entry point in the subsystem is allowed. At least one job entry point of a subsystem must be defined with FIRST-CONNECTION=*ALLOWED.

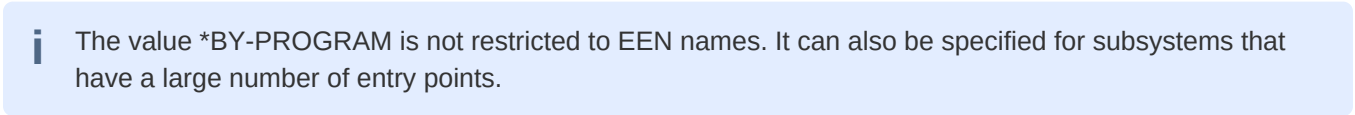
FIRST-CONNECTION = *ALLOWED
Default value: first connection to the specified job entry point is allowed.

FIRST-CONNECTION = *FORBIDDEN
Connection to the specified job entry point via SVC or ISL is not allowed if the task has not yet been connected to another job entry point belonging to the subsystem.
It is not permitted to specify this operand value for job entry points that have been defined with MODE=*LINK/*SYSTEM-EXIT or CONNECTION-ACCESS=*SIH.

SUBSYSTEM-ENTRIES = *BY-PROGRAM(...)
The entry points of the specified subsystem are supplied dynamically from the BLS name list at load time instead of statically from the catalog. A prerequisite for this functionality is the use of BLSSERV as of version 2.1, that supports using Extended External Names (EEN names) as entry points for DSSM subsystems.

The value cannot be specified together with the SUBSYSTEM-ACCESS=*SYSTEM operand.

The MODE, CONNECTION-ACCESS and FIRST-CONNECTION operands are supplied with the default values if the entry points are defined with this value.


The value *BY-PROGRAM is not restricted to EEN names. It can also be specified for subsystems that have a large number of entry points.

CONNECTION-SCOPE = *TASK / *PROGRAM
The connection is shut down at task or program termination.

MEMORY-CLASS =
Specifies the subsystem-specific address space in which the subsystem is to be loaded. System administration can use this operand to define the address space valid for the subsystem concerned so as to meet the special requirements of the installation.

MEMORY-CLASS = *SYSTEM-GLOBAL(...)
Default value: the subsystem will be loaded in class 3 or class 4 memory. Resident CSECTs will be given class 3 memory, all others will be given pageable class 4 memory.

SUBSYSTEM-ACCESS =
Identifies the access authorization (privileges) and location of the requested memory.

SUBSYSTEM-ACCESS = *LOW
Default value: nonprivileged address space below the 16-Mbyte boundary is allocated.

SUBSYSTEM-ACCESS = *SYSTEM
Subsystems declared with this operand value are privileged subsystems to which privileged address space above the 16-Mbyte boundary is allocated.
This operand value is mandatory for subsystems whose entry point is declared via SVC (MODE=*SVC) or for which an INIT, STOPCOM, DEINIT or CLOSE-CTRL routine is declared. It is not permitted with CONNECTION-ACCESS=*ALL and MODE=*LINK. The value cannot be specified together with an entry point that was specified together with CONNECTION-ACCESS=*ALL or MODE=*LINK.
The value cannot be specified together with the SUBSYSTEM-ENTRIES=*BY-PROGRAM operand.

SUBSYSTEM-ACCESS = *HIGH
Nonprivileged address space up to 2 Gbytes is allocated.

MEMORY-CLASS = *LOCAL-PRIVILEGED(...)
The subsystem is given a memory pool in nonprivileged class 5 memory, located below the 16-Mbyte boundary.

This specification is suitable for nonprivileged subsystems which demand a relatively large amount of address space (approx. 1 Mbyte) and have to be set up below the 16-Mbyte boundary. These subsystems are loaded in memory pools at the same address, in order to manage the use of the limited address space below 16 Mbytes. Although such subsystems are loaded in parallel in the same address space, they cannot be used simultaneously by the same task (see also the SEPARATE-ADDRESS-SPACE statement).
The subsystem must not contain any resident CSECTs, as otherwise a later attempt to activate it will be aborted.

SIZE = <integer 1..32767>
Size of the required address space (in 4Kbyte pages) for the memory pool in class 5 memory. This value should be set at least high enough to ensure that the subsystem and any selectable units/load units which it may load dynamically can be loaded in their entirety. The upper limit is generation-specific.

MEMORY-CLASS = *LOCAL-UNPRIVILEGED(...)
The subsystem is given a memory pool in nonprivileged class 6 memory. This specification is reserved for subsystems which can be executed like a program.
In keeping with this, their access authorization (privileges) must be defined with the value *ALL in the CONNECTION-ACCESS operand.
This operand value must not be specified together with an entry point which was defined with CONNECTION-ACCESS=*SYSTEM.

The subsystem must not contain any resident CSECTs, as otherwise a later attempt to activate it will be aborted. If this operand value is specified, only CONNECTION-SCOPE=*PROGRAM is permissible for clearing the connection to the specified subsystem entry point.

SIZE = <integer 1..32767>
Size of the required address space (in 4Kbyte pages) for the memory pool in class 6 memory. This value should be set at least high enough to ensure that the subsystem and any selectable units/load units which it may load dynamically can be loaded in their entirety. The upper limit is generation-specific.

SUBSYSTEM-ACCESS =
Identifies the location of the requested memory space.

SUBSYSTEM-ACCESS = *LOW
Default value: nonprivileged address space below the 16-Mbyte boundary is allocated.Because this specification is suitable for subsystems which can be executed like programs, it is advisable additionally to specify CONNECTION-SCOPE=*PROGRAM.

SUBSYSTEM-ACCESS = *HIGH
Nonprivileged address space up to 2 Gbytes is allocated.

Because this specification is suitable for subsystems which can be executed like programs, it is advisable additionally to specify CONNECTION-SCOPE=*PROGRAM.

START-ADDRESS =
Defines the start address in class 6 memory.

START-ADDRESS = *ANY
Default value: the location of the subsystem in class 6 memory will be determined by DSSM.

START-ADDRESS = <x-string 7..8>
Start address in the segment raster at which the subsystem’s start address is to be located. The value specified must be an 8-character hexadecimal constant which is a multiple of X'100000'.

MEMORY-CLASS = *BY-SLICE(...)
The specified subsystem is a nonprivileged subsystem and consists of an LLM, which in turn consists of a shareable code (program area) and a non-shareable code (data area). The program area is loaded into the shareable address space (this corresponds to MEMORY-CLASS=*SYSTEM-GLOBAL). The data area is loaded into the user address space of the holder task and is copied into the private user address spaces of the connected tasks at the same address.
The following values must be specified together with MEMORY-CLASS=*BY-SLICE:
SUBSYSTEM-LOAD-MODE=*ADVANCED and CONNECTION-ACCESS=*ALL.
The value can only be specified together with the MODE=*LINK and CONNECTION-SCOPE=*TASK or *PROGRAM operands.

SIZE = <integer 1.32767>
Specifies the size of the requested memory space for the data area in 4K pages. The value chosen here must be sufficiently large to allow the subsystem and, if appropriate, selectable units/load units dynamically loaded by the subsystem to be loaded in full. The upper limit is dependent on generation.

LINK-ENTRY = <text 1..8 without-sep>(...)
Defines the name of the object module/ENTRY/CSECT required for loading (for the operand in the call of the $PBBND1 macro to the dynamic binder loader DBL). The subsystem must be completely loaded by this ENTRY (if necessary, using autolink).

AUTOLINK =
Controls invocation of the autolink function during linking and loading.
The linkage editor's autolink function permits the automatic insertion of modules which are not explicitly inserted by appropriate statements. The main purpose of this function is to save users of higher-level programming languages from having to make explicit statements to insert individually the numerous modules of the runtime system which are required. Further details of the autolink function will be found in the "BLSSERV" manual.

The autolink function can also be implicitly circumvented if the first external reference encountered during linkage editing of the object module which is to be loaded points to a prelinked module. The advantage of this approach is that the paging behavior when the subsystem is later executed can be optimized at this preliminary stage (during linkage editing). In addition, errors during linkage editing can be avoided in this way.

AUTOLINK = *FORBIDDEN /*ALLOWED
Default value: the autolink function will be suppressed or allowed.

REFERENCED-SUBSYSTEM =
Specifies whether there is a list of subsystems to which there are address relations, and which is to be used for resolving external references

REFERENCED-SUBSYSTEM = *NONE
Default value: the subsystem which is being defined has no external references.

REFERENCED-SUBSYSTEM = list-poss(15): <structured-name 1..8>
The system which is being defined has external references to a maximum of 15 other subsystems; these subsystems must be used in resolving the external references. If any of the subsystems nominated here is missing at the time of activation or deactivation (and if a check of the external references has also been requested by the operand CHECK-REFERENCE=*YES), the action will be aborted.
It is also possible to address the BS2000 control program via these external references - using the name CP. In the substructure which follows, it is possible to specify either exactly one version, or a range of versions within which all versions are to be referred to.
If a version range list is used to limit the version of the CP subsystem, DSSM checks the compatibility of the current CP version against the versions in the range list. The subsystem will only be loaded if it is a compatible version.

The following restrictions should be noted when specifying subsystems to which there are address relations:

- No address relations may be declared to subsystems which have the attribute MEMORY-CLASS=*LOCAL-PRIVILEGED/*LOCAL-UNPRIVILEGED/*BY-SLICE.
- Subsystems which have the attribute STOP-AT-SHUTDOWN=*NO may have address relations only to other subsystems which also have this attribute.
- If the attribute SUBSYSTEM-ACCESS=*SYSTEM is specified for the subsystem being defined, subsystems defined with SUBSYSTEM-ACCESS=*LOW or SUBSYSTEM-ACCESS=*HIGH must not be addressed.
- As a rule, a nonprivileged subsystem must not have any address relations to the control program (CP).
- If a reference is made to a subsystem which has at least one version that may be operated in coexistence or exchange mode, a unique version must be specified.
- Any address relations must be defined in accordance with the start attributes (CREATION-TIME operand); i.e. a subsystem may only have relations to other subsystems if these are started at the same time or earlier.

UNRESOLVED-EXTERNALS =
Defines how the load procedure is to behave if there are unresolved external references.

UNRESOLVED-EXTERNALS = *FORBIDDEN
Default value: if unresolved external references occur, the load procedure will be terminated.

UNRESOLVED-EXTERNALS = *ALLOWED
The load procedure will be continued, unresolved external references will be given the value X'FFFFFFFF'.

CHECK-REFERENCE =
Defines whether or not the subsystems specified in the REFERENCED-SUBSYSTEM operand are to be checked in respect of their status and availability.

CHECK-REFERENCE = *YES
Default value: the referenced subsystems will be checked. If any of them is missing, DSSM will abandon the activation or unloading of the subsystem.

CHECK-REFERENCE = *NO
DSSM is not to carry out any check. However, even if the user generates complex subsystems with this statement, DSSM will still perform the requested functions **in spite of** the risk of conflicts:

- the START-SUBSYSTEM command will load the specified subsystem, even if there is a subsystem to which it has defined relations and which is not yet fully loaded;
- the commands RESUME-SUBSYSTEM, STOP-SUBSYSTEM and HOLD-SUBSYSTEM will be executed by DSSM without any checks being made on relations and dependencies.

RELATED-SUBSYSTEM =
Defines whether or not there is a list of subsystems for which there are dependency relations.

RELATED-SUBSYSTEM = *NONE
Default value: there are no dependency relations for the subsystem which is being defined.

RELATED-SUBSYSTEM = list-poss(100): <structured-name 1..8>
The subsystem which is being defined has dependency relations to up to 100 other subsystems, without which the subsystem currently being defined cannot function.

It is also permissible to define dependency relations to the BS2000 control program (CP). The rules and restrictions which apply when doing so are analogous to those for address relations, where they are described in more detail (REFERENCED-SUBSYSTEM operand).
A dependency relation always points to a single version of a subsystem. In the substructure which follows, it is possible to specify either exactly one version, or a range of versions within which all versions are to be referred to.

The following general restrictions should be noted when specifying dependent subsystems:

- The relation which is defined must not contain closed loops. A loop arises if subsystem A is dependent on B, B is dependent on C and this is in turn dependent on A.
- If the subsystem which is being defined has been given the attribute MEMORY-CLASS=*SYSTEM-GLOBAL, it is not permissible to address any subsystems defined with MEMORY-CLASS=*LOCAL-PRIVILEGED or *LOCAL-UNPRIVILEGED.
- For subsystems which have the attribute SUBSYSTEM-ACCESS=*SYSTEM, no dependency relations may be defined to subsystems to which SUBSYSTEM-ACCESS=*LOW or SUBSYSTEM-ACCESS=*HIGH or MEMORY-CLASS=*BY-SLICE applies.
- The dependency relations must be defined to correspond to the start attributes (CREATION-TIME operand); i.e. a subsystem may only be dependent on subsystems which are started at the same time or earlier.

LOWEST-VERSION =
Specifies the lowest value (lowest version) in the subsystem version range list.

LOWEST-VERSION = *LOWEST-EXISTING
The lowest version in the catalog is to be addressed.

LOWEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the lower limit of the range of versions.

HIGHEST-VERSION =
Specifies the upper value (highest version) in the subsystem version range list.

HIGHEST-VERSION = *HIGHEST-EXISTING
The highest version in the catalog is to be addressed.

HIGHEST-VERSION = <c-string 3..8> / <text 3..8>
Version of the subsystem which is to be used as the upper limit of the range of versions.

4.3.14 SHOW-CATALOG

Show subsystem configuration

Function

This statement can be used to output the contents of a subsystem configuration, which is stored in a subsystem catalog, to either the screen or a file, as required.
The output always contains the name of the subsystem catalog and information on whether and how the catalog is saved in the event of an error (this behavior is set in the statement SAVE-CATALOG ..., FORCED=...).

SHOW-CATALOG will be rejected if the specified file or subsystem name or the specified subsystem version does not exist. If the subsystem catalog is empty, a warning message will be issued and the statement aborted.

If the specified file name does not correspond to that of the catalog which is currently open, the following message is output:

```
SCM0011   DO YOU REALLY WANT TO OVERWRITE MEMORY CATALOG '(&00)'? REPLY (Y/N)
```

If the reply is **Y**, the virtual definitions in the current catalog will be lost.

If the reply is **N**, execution of the SHOW-CATALOG statement will be aborted. With the aid of SAVE-CATALOG, the user can store in a file all subsystem definitions not yet saved.

If the CHECK-CATALOG statement was not used to run a consistency check beforehand, SHOW-CATALOG may detect consistency errors in the subsystem catalog.

Format

SHOW-CATALOG

CATALOG-NAME = ***CURR**ENT / <filename 1..54 without-gen-vers>

,**CONTAINED-SUBSYSTEMS** = ***NO** / ***Y**ES

,**SUBSYS**TEM-NAME = ***ALL** / <structured-name 1..8>(...)

<structured-name 1..8>(...)

VERSION = ***ALL** / <c-string 3..8> / <text 3..8>

,**GENERAL-ATTRIBUTES** = ***Y**ES / ***NO**

,**INTERNAL-ENTRIES** = ***Y**ES / ***NO**

,**MEMORY-ATTRIBUTES** = ***Y**ES / ***NO**

,**RELATED-FILES** = ***Y**ES / ***NO**

,**LINK-ATTRIBUTES** = ***Y**ES / ***NO**

,**REFERENCE-RELATION** = ***Y**ES / ***NO**

,**DEPENDENCE-RELATION** = ***Y**ES / ***NO**

,**ADDR-SPACE-RELATION** = ***Y**ES / ***NO**

,**HOLDER-TASK-INFO** = ***Y**ES / ***NO**

,**SUBSYSTEM-ENTRIES** = ***Y**ES / ***NO**

,**OUTPUT** = ***SYSLST** (...) / ***SYSOUT**

***SYSLST**(...)

SYSLST-NUMBER = ***STD** / <integer 1..99>

,**SUMM**ARY = ***Y**ES / ***NO**

Operands

CATALOG-NAME =
Specifies the name of the catalog which contains the definitions that are to be displayed. If this catalog file name does not exist or if the specified catalog is empty, the statement will be rejected.

CATALOG-NAME = *CURRENT
Default value: the contents of the catalog which is currently open (START-CATALOG-CREATION or START-CATALOG-MODIFICATION statement) are to be output.

CATALOG-NAME = <filename 1..54 without-gen-vers>
Fully qualified name of the static subsystem catalog whose contents are to be displayed.

CONTAINED-SUBSYSTEMS = *NO / *YES
Specifies whether to output the list of DSSM subsystems defined in the catalog (*YES) or not (*NO).

SUBSYSTEM-NAME =
Specifies the subsystems on which information is being requested.
If the name of the subsystem or a version on which information is requested does not exist, the statement will be rejected. The scope of the information to be output for the subsystems can be restricted by specifying appropriate criteria in the following operands.

SUBSYSTEM-NAME = *ALL
Default value: information is requested about all the subsystems which are listed in the catalog directory.

SUBSYSTEM-NAME = <structured-name 1..8>(...)
Name of the subsystem for which SSCM is to provide information from the catalog.

VERSION =
Specifies the version of the selected subsystem.

VERSION = *ALL
The information which is output should cover all versions of the subsystem stored in the catalog.

VERSION = <c-string 3..8> / <text 3..8>
Information from the catalog should only be provided for these versions of the selected subsystem.

GENERAL-ATTRIBUTES = *YES / *NO
Specifies whether the following general attributes of the named subsystems are to be read from the catalog (*YES) or not (*NO):

- when is the subsystem to be started after system initialization?
  (CREATION-TIME)
- in what mode is the subsystem to be loaded?
  (SUBSYSTEM-LOAD-MODE)
- is the subsystem to be automatically unloaded at shutdown?
  (STOP-AT-SHUTDOWN)
- may the subsystem be suspended or unloaded after it has been loaded?
  (SUBSYSTEM-HOLD)
- may the commands for controlling a subsystem be used?
  (STATE-CHANGE-CMDS)
- is the FORCE option permitted?
  (FORCED-STATE-CHANGE)
- is the RESET option permitted?
  (RESET)
- must the initialization routine be executed if the holder task is terminated abnormally?
  (RESTART-REQUIRED)
- may more than one version of the subsystem be active simultaneously?
  (VERSION-COEXISTENCE)
- may two versions of a subsystem be dynamically exchanged?
  (VERSION-EXCHANGE)
- What is the name of the installation unit of the subsystem?
  (INSTALLATION-UNIT)
- What is the copyright (text and date) of the subsystem?
  (COPYRIGHT)

INTERNAL-ENTRIES = *YES / *NO
Specifies whether SSCM is to provide the following information about the entry points to the specified subsystems (*YES) or not (*NO):

- the names of the entry points for the subsystem routines INIT, STOPCOM, DEINIT and CLOSE-CTRL

- the name of the entry point to be used for dynamic identity checking (DYNAMIC-CHECK-ENTRY)
- the name of the interface version to be used in calling the INIT, STOPCOM, DEINIT or CLOSE-CTRL routine (INTERFACE-VERSION).

MEMORY-ATTRIBUTES = *YES / *NO
Specifies whether the following memory-related information on the subsystems, which is stored in the catalog, is to be output (*YES) or not (*NO):

- memory class (MEMORY-CLASS)
- size of required address space (SIZE)
- start address of the subsystem code (START-ADDRESS)
- privileges and access authorization for the address space (SUBSYSTEM-ACCESS)

RELATED-FILES = *YES / *NO
Specifies whether to supply information about the subsystem satellites (*YES) or not (*NO). This output includes information as to whether it is mandatory to use a REP file for this subsystem (REP-FILE-MANDATORY) and concerning the user ID under which the satellites are cataloged (INSTALLATION-USERID).

The term “subsystem satellites” covers:

- the subsystem’s object module file (LIBRARY)
- the message file (MESSAGE-FILE)
- the syntax file (SYNTAX-FILE)
- the subsystem’s information file (SUBSYSTEM-INFO-FILE)
- the REP file (REP-FILE)

LINK-ATTRIBUTES = *YES / *NO
Specifies whether the information on the linkage and loading of the subsystem is to be read from the catalog (*YES) or not (*NO):

- the name of the object module/ENTRY/CSECT required for loading (LINK-ENTRY)
- the inclusion of the autolink function (AUTOLINK)
- the information on system response when there are unresolved external references (UNRESOLVED)
- the inclusion of a check run for referenced subsystems (CHECK-REFERENCE)

REFERENCE-RELATION = *YES / *NO
Specifies whether the list of subsystems to which address relations exist is to be included in the output of catalog information (*YES) or not (*NO).

DEPENDENCE-RELATION = *YES / *NO
Specifies whether the list of subsystems to which dependency relations exist is to be included in the output of catalog information (*YES) or not (*NO).

ADDR-SPACE-RELATION = *YES / *NO
Specifies whether the list of subsystems with which any address space overlap must be avoided is to be included in the output of catalog information (*YES) or not (*NO).

HOLDER-TASK-INFO = *YES / *NO
Specifies whether the identity of the holder task and the list of subsystems which are to be created within a common holder task is to be included in the output of catalog information (*YES) or not (*NO).

SUBSYSTEM-ENTRIES = *YES / *NO
Specifies whether the list of entry points declared in the definition of the subsystem and the following attributes of these entries are to be read from the catalog (*YES) or not (*NO):

- type of the declared job entry point (MODE)
- number of the routine (for an SVC or system exit) (NUMBER)
- the function number of the entry point (FUNCTION-NUMBER)
- the version of the function number (FUNCTION-VERSION)
- information about calling via system exit routines (CALL-BY-SYSTEM-EXIT)
- the privileges and access authorization for entry points (CONNECTION-ACCESS and CONNECTION-SCOPE)

OUTPUT =
Specifies where the information generated by the statement is to be output.

OUTPUT = *SYSLST(...)
Default value: the messages are to be output to SYSLST.

SYSLST-NUMBER =
Identifies the SYSLST file to which the output is to be directed.

SYSLST-NUMBER = *STD
Default value: output should go to the standard system file, SYSLST.

SYSLST-NUMBER = <integer 1..99>
Output is to go to one of the system files from the set SYSLST01 to SYSLST99, the number of which is given here.

SUMMARY = *YES / *NO
Specifies whether the requested output is to be preceded by a summary of all the subsystems listed in the catalog directory (*YES) or not (*NO).
For OUTPUT=*SYSLST(SUMMARY=*YES) see the overview of abbreviations below.

OUTPUT = *SYSOUT
The messages will be output to the data display terminal.

Overviews of the subsystems in the catalog (see operand OUTPUT=*SYSLST(SUMMARY=*YES)) make use of the following abbreviations:

- for CREATION-TIME
  - ACR: *AT-CREATION-REQUEST
  - ASC: *AT-SUBSYSTEM-CALL
  - ADL: *AT-DSSM-LOAD
  - BDL: *BEFORE-DSSM-LOAD
  - MAS: *MANDATORY-AT-STARTUP
  - BSR: *BEFORE-SYSTEM-READY
  - ASR: *AFTER-SYSTEM-READY

- for MEMORY-CLASS
  - S: *SYSTEM-GLOBAL
  - P: *LOCAL-PRIVILEGED
  - U: *LOCAL-UNPRIVILEGED
  - B: *BY-SLICE
- for SUBSYSTEM-ACCESS
  - SYS: *SYSTEM
  - ALL:: *LOW / *HIGH
- for INTERNAL-ENTRIES
  - DYN: *DYNAMIC
  - YES: name
  - NO: *NO
- for CONNECTION-ACCESS
  - SYS: *SYSTEM
- for STATE-CHANGE-CMDS
  - ADM: *BY-ADMINISTRATOR-ONLY
- for REP-FILE
  - MAN: REP-FILE = *STD / filename and REP-FILE-MANDATORY = *YES
- for GENERAL-ATTRIBUTES
  - YES: *ALLOWED
  - NO: *FORBIDDEN
- für RELATED-FILES
  - YES: *STD / filename
  - NO: *NO
  - IMO: *INSTALLED

4.3.15 SHOW-SSD

Show contents of SSD object (subsystem definitions)

Function

This statement outputs the contents of an SSD object either to the screen or to another file. The definitions of one or more subsystems are stored in an SSD object.
Each of the subsystem definitions which is stored in the specified SSD object (ISAM file) will be output separately.

The name and version of the SSD object are always output, as is the name of the domain for the SSD object and information on whether PULS problem messages have been incorporated in the SSD object.

It should be borne in mind that SHOW-SSD does not output the complete contents of the SSD object, but only the subsystem definition statements entered since the last ADD-CATALOG-ENTRY statement.

Format

SHOW-SSD
SSD-FILE-NAME = ***CURR**ENT / <filename 1..54 without-gen-vers>
,**GENERAL-ATTRIBUTES** = ***Y**ES / ***NO**
,**INTERNAL-ENTRIES** = ***Y**ES / ***NO**
,**MEMORY-ATTRIBUTES** = ***Y**ES / ***NO**
,**RELATED-FILES** = ***Y**ES / ***NO**
,**LINK-ATTRIBUTES** = ***Y**ES / ***NO**
,**REFERENCE-RELATION** = ***Y**ES / ***NO**
,**DEPENDENCE-RELATION** = ***Y**ES / ***NO**
,**ADDR-SPACE-RELATION** = ***Y**ES / ***NO**
,**HOLDER-TASK-INFO** = ***Y**ES / ***NO**
,**SUBSYSTEM-ENTRIES** = ***Y**ES / ***NO**
,**OUTPUT** = ***SYSLST** (...) / ***SYSOUT**
***SYSLST**(...)
SYSLST-NUMBER = ***STD** / <integer 1..99>

Operands

SSD-FILE-NAME =
Specifies the name of the SSD object (ISAM file) which contains the definitions that are to be displayed. If there is no ISAM file with this name or if the specified file is empty, the statement will be rejected.

SSD-FILE-NAME = *CURRENT
Default value: the contents of the SSD object which is currently open (START-SSD-CREATION statement) are to be output.

SSD-FILE-NAME = <filename 1..54 without-gen-vers>
Fully qualified name of the SSD object whose contents are to be displayed.

GENERAL-ATTRIBUTES = *YES / *NO
Specifies whether the following general attributes of the subsystems defined in the SSD object are to be displayed (*YES) or not (*NO):

- when is the subsystem to be started after system initialization?
  (CREATION-TIME)
- in which load mode is the subsystem to be loaded?
  (SUBSYSTEM-LOAD-MODE)
- is the subsystem to be automatically unloaded at shutdown?
  (STOP-AT-SHUTDOWN)
- may the subsystem be suspended or unloaded after it has been loaded?
  (SUBSYSTEM-HOLD)
- may the commands for controlling a subsystem be used?
  (STATE-CHANGE-CMDS)
- is the FORCE option permitted?
  (FORCED-STATE-CHANGE)
- is the RESET option permitted?
  (RESET)
- must the initialization routine be executed if the holder task is terminated abnormally?
  (RESTART-REQUIRED)
- may more than one version of the subsystem be active simultaneously?
  (VERSION-COEXISTENCE)
- may two versions of a subsystem be dynamically exchanged?
  (VERSION-EXCHANGE)
- What is the name of the installation unit of the subsystem?
  (INSTALLATION-UNIT)
- What is the copyright (text and date) of the subsystem?
  (COPYRIGHT)

INTERNAL-ENTRIES = *YES / *NO
Specifies whether SSCM is to provide the following information about the entry points to the subsystems contained in the file (*YES) or not (*NO):

- the names of the entry points for the subsystem routines INIT, STOPCOM, DEINIT and CLOSE-CTRL
- the name of the entry point to be used for dynamic identity checking (DYNAMIC-CHECK-ENTRY)
- the name of the interface version to be used in calling the INIT, STOPCOM, DEINIT or CLOSE-CTRL routine (INTERFACE-VERSION).

MEMORY-ATTRIBUTES = *YES / *NO
Specifies whether the following memory-related information on the subsystems, which is stored in the SSD object as part of the subsystem definition, is to be output (*YES) or not (*NO):

- memory class (MEMORY-CLASS)
- size of the required address space (SIZE)

- start address of the subsystem code (START-ADDRESS)
- privileges and access authorization for the address space (SUBSYSTEM-ACCESS)

RELATED-FILES = *YES / *NO
Specifies whether information about the subsystem satellites is to be supplied (*YES) or not (*NO). This output includes information as to whether it is mandatory to use a REP file for this subsystem (REP-FILE-MANDATORY) and concerning the user ID under which the satellites are cataloged (INSTALLATION-USERID). The term “subsystem satellites” covers:

- the subsystem’s object module file (LIBRARY)
- the message file (MESSAGE-FILE)
- the syntax file (SYNTAX-FILE)
- the subsystem’s information file (SUBSYSTEM-INFO-FILE)
- the REP file (REP-FILE)

LINK-ATTRIBUTES = *YES / *NO
Specifies whether the information on the linkage and loading of the subsystem is to be read from the SSD object (*YES) or not (*NO):

- the name of the object module/ENTRY/CSECT required for loading (LINK-ENTRY)
- the inclusion of the autolink function (AUTOLINK)
- the information on system response when there are unresolved external references (UNRESOLVED)
- the inclusion of a check run for referenced subsystems (CHECK-REFERENCE)

REFERENCE-RELATION = *YES / *NO
Specifies whether the list of subsystems to which address relations exist is to be included in the output of SSD file information (*YES) or not (*NO).

DEPENDENCE-RELATION = *YES / *NO
Specifies whether the list of subsystems to which dependency relations exist is to be included in the output of SSD object information (*YES) or not (*NO).

ADDR-SPACE-RELATION = *YES / *NO
Specifies whether the list of subsystems with which any address space overlap must be avoided is to be included in the output of SSD file information (*YES) or not (*NO).

HOLDER-TASK-INFO = *YES / *NO
Specifies whether the identity of the holder task and the list of subsystems which are to be created within a common holder task are to be included in the output of SSD file information (*YES) or not (*NO).

SUBSYSTEM-ENTRIES = *YES / *NO
Specifies whether the list of entry points declared in the definition of the subsystem and the following attributes of these entries are to be read from the SSD object (*YES) or not (*NO):

- type of the declared entry point (MODE)
- number of the routine (for an SVC or system exit) (NUMBER)
- function number of the entry point (FUNCTION-NUMBER)
- version of the function number (FUNCTION-VERSION)
- information about invocation via system exit routines (CALL-BY-SYSTEM-EXIT)

- the privileges and access authorization for entry points
  (CONNECTION-ACCESS and CONNECTION-SCOPE)

OUTPUT =
Specifies where the information generated by the statement is to be output.

OUTPUT = *SYSLST(...)
Default value: the messages are to be output to SYSLST.

> **SYSLST-NUMBER =**
> Identifies the SYSLST file to which the output is to be directed.
>
> **SYSLST-NUMBER = *STD**
> Default value: output is to go to the standard system file, SYSLST.
>
> **SYSLST-NUMBER = <integer 1..99>**
> Output is to go to one of the system files from the set SYSLST01 to SYSLST99, the number of which must be given here.

OUTPUT = SYSOUT
The messages will be output to the data display terminal.

4.3.16 START-CATALOG-CREATION

Define name of static subsystem catalog

Function

This statement declares the name of a new static subsystem catalog. The SSD objects containing the definitions of the subsystems can then be inserted in this file.

START-CATALOG-CREATION will be rejected and an error message issued if a file of the same name already exists, or if the same catalog entry has been created using CREATE-FILE. The definition is terminated by saving the new catalog (SAVE-CATALOG statement).

However, if there are two consecutive START-CATALOG-CREATION or START-CATALOG-MODIFICATION statements, the following message will appear in interactive jobs:

```
SCM0011   DO YOU REALLY WANT TO OVERWRITE MEMORY CATALOG '(&00)'? REPLY (Y/N)
```

If the reply is **Y**, the reserved memory space will be released, and the declared attributes of the catalog will be abandoned.
If the reply is **N**, the second START-CATALOG-CREATION statement is considered invalid and the first statement can be terminated by SAVE-CATALOG.

In the case of batch jobs, the response **Y** is given implicitly.

Format

START-CATALOG-CREATION
CATALOG-NAME = **<u>*STD</u>** / <filename 1..51 without-gen-vers>

Operands

CATALOG-NAME =
Name of the static subsystem catalog which is to be created.

If no catalog of this name exists, the statement will be rejected.

CATALOG-NAME = <u>*STD</u>
The default value is the file SYS.SSD.CAT.X on the home pubset.

CATALOG-NAME = <filename 1..51 without-gen-vers>
Fully qualified file name.

4.3.17 START-CATALOG-MODIFICATION

Modify static subsystem catalog

Function

This statement allows an existing static subsystem catalog to be modified. New SSD objects containing the definitions of the subsystems can then be inserted in this file.

The definition of a modified catalog is terminated by means of the statements CHECK-CATALOG statement and SAVE-CATALOG statement .

START-CATALOG-MODIFICATION will be rejected if the name specified for the file does not exist, or was not created beforehand by means of a START-CATALOG-CREATION statement.

If there are two consecutive START-CATALOG-MODIFICATION statements or if START-CATALOG-MODIFICATION is specified after the statement START-CATALOG-CREATION without a concluding SAVE-CATALOG statement, the following message will be displayed in interactive jobs:

```
SCM0011  DO YOU REALLY WANT TO OVERWRITE MEMORY CATALOG '(&00)'? REPLY (Y/N)
```

If the reply is **Y**, the reserved memory space will be released, and the changes just made to attributes of the catalog will be abandoned.
If the reply is **N**, the second START-CATALOG-MODIFICATION statement is considered invalid, and the first statement can be concluded with SAVE-CATALOG.

In the case of batch jobs, the response **Y** is given implicitly.

Format

START-CATALOG-MODIFICATION
CATALOG-NAME = <filename 1..54 without-gen-vers>

Operands

CATALOG-NAME = <filename 1..54 without-gen-vers>
Fully qualified file name of the subsystem catalog which is to be modified.

4.3.18 START-SSD-CREATION

Generate SSD object for adding subsystem definitions

Function

The START-SSD-CREATION statement initiates the generation of an SSD object for adding subsystem definitions. The user must specify the name of the SSD object, the name of the file in which it is to be stored, and the names of the files required or referenced by the subsystem.

START-SSD-CREATION is rejected and an error message issued if the file to be specified for SSD-FILE-NAME already exists.

The SAVE-CATALOG statement for saving all declared attributes can be entered only after successfully executing the SET-SUBSYSTEM-ATTRIBUTES statement.

Format

START-SSD-CREATION
SSD-NAME = <name 1..8> ,**VERSION** = <integer 1..999> ,**DOMAIN** = <structured-name 1..13> ,**CORRECTION** = ***NONE** / list-poss(20): <alphanum-name 8..8> ,**SSD-FILE-NAME** = <filename 1..51 without-userid without-gen-vers> ,**BLOCK-CONTROL-INFO** = ***STD** / ***WITHIN-DATA-BLOCK**

Operands

SSD-NAME = <name 1..8>
Name of the SSD object.

The name must comply with the following convention: the first three letters correspond to the product's message class, and the last three letters are "SSC".

VERSION = <integer 1.999>
Version of the SSD object.

DOMAIN = <structured-name 1..13>
Identifier of the MONSYS domain for the SSD object. The name of this domain is used internally by the system to implement the unique, consistent assignment of all components to the subsystem.

CORRECTION =
Indication of whether PULS problem messages are incorporated in the SSD object for the subsystems.

CORRECTION = *NONE
Default value: no PULS problem messages are incorporated in the SSD object.

CORRECTION = list-poss(20) <alphanum-name 8..8>
List of a maximum of 20 problem messages which are corrected in the SSD object.

SSD-FILE-NAME = <filename 1..51 without-userid without-gen-vers>
Name of the new ISAM file which is to be created and in which the SSD object is to be saved. If the file name already exists, the statement will be rejected.

BLOCK-CONTROL-INFO =
Specifies the file format in which the file for the SSD object is to be created.

BLOCK-CONTROL-INFO = *STD
Default setting: SSCM first attempts to create the SSD object as a K file (block format PAMKEY) with block length 1 (BUFFER-LENGTH=*STD(SIZE=1)).
In the event of an error an attempt is made to create the SSD object with block length 2.
If this attempt is also unsuccessful, the SSD object is created as an NK file with block format DATA and block length 2.

BLOCK-CONTROL-INFO = *WITHIN-DATA-BLOCK
SSCM creates the SSD object as an NK file in the DATA block format and with block length 2.

Notes

Successful processing of the START-SSD-CREATION statement can be followed only by the statements listed below. The ones marked with an asterisk can be used more than once in order to define different subsystems (but not different versions of the same subsystem):

- ADD-SUBSYSTEM-ENTRIES (*)
- ASSIGN-HOLDER-TASK (*)
- SEPARATE-ADDRESS-SPACE (*)
- SET-SUBSYSTEM-ATTRIBUTES (*)
- SHOW-SSD

4.4 Installing SSCM

It is recommended to install SSCM with IMON.

SSCM V21.0 is delivered with the following files:

- the module library SYSLNK.SSCM.210, which contains the SSCM object module
- the system syntax file SYSSDF.SSCM.210 containing statements for SSCM
- the user syntax file SYSSDF.SSCM.210.USER containing statements for SSCM
- the procedure SYSPRC.SSCM.210, which SSCM starts by means of START-EXECUTABLE-PROGRAM
- the subsystem declaration file SYSSSC.SSCM.210
- the message file SYSMES.SSCM.210
- the REP file SYSREP.SSCM.210

Before you can install SSCM V21.0, it is first necessary to satisfy the following conditions:

1. The SSCM subsystem definition must be entered in the subsystem catalog.
2. The SYSLNK.SSCM.210 library must be cataloged under the installation user ID that was named with ADD-CATALOG-ENTRY or SET-SUBSYSTEM-ATTRIBUTES when SSCM was entered in the subsystem catalog (by default, this is the system user ID whose files generally begin with “$.”).
3. The SSCM subsystem is activated by means of the command START-SUBSYSTEM SSCM and it is started by means of the command START-SSCM.
   This subsystem also includes the message file, the syntax file and the REP file, which are activated automatically. The system syntax file SYSSDF.SSCM.023 must have the attribute SHARE=SPECIAL.

If the procedure SYSPRC.SSCM.210 is used to call SSCM V21.0, points 1 and 3 cease to apply.

Coexistence of SSCM versions

SSCM versions V1.0 to V21.0 can be defined simultaneously in the system catalog. The version can be selected with the SELECT-PRODUCT-VERSION command. The START-SSCM command then starts the selected version.

4.5 Examples

The following examples show sequences of SSCM statements that are intended to facilitate understanding of the procedure for creating and modifying certain objects.

4.5.1 Generation of an SSD object

```
/START-SSCM
//START-SSD-CREATION SSD-NAME=SS1SSC,VERSION=001,DOMAIN=DSSM,
        CORRECTION=*NONE,SSD-FILE-NAME=SYSSSC.SS1.001 ————————————————  (1)
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=SS1(VERSION=1.0),
        SUBSYSTEM-ENTRIES=ENTRY1(CONNECTION-ACCESS=*ALL,
                                 CONNECTION-SCOPE=*TASK),
        REFERENCED-SUBSYSTEM=SS2(LOWEST-VERS=01.0,HIGHEST-VERSION=02.5),
        RELATED-SUBSYSTEM=SS3(LOWEST-VERSION=02.0,HIGHEST-VERSION=02.0),
        LINK-ENTRY=E1 ————————————————————————————————————————————————  (2)
//SEPARATE-ADDRESS-SPACE SUBSYSTEM-NAME=SS1,
        FROM-SUBSYSTEMS=(SS5,SS6,SS9) ————————————————————————————————  (3)
//ASSIGN-HOLDER-TASK TYPE=*SHARED-HOLDER(
        BY-SUBSYSTEMS=(SS1,SS2,SS3),TSN=*BY-DSSM) ————————————————————  (4)
//SAVE-SSD ——————————————————————————————————————————————————————————  (5)
```

(1) Declaration of the SSD object.

(2) Definition of the chief attributes of the SS1 subsystem: the entry points, references and dependencies.

(3) Specification of the subsystems which must not share address space with SS1; in this example, these are the subsystems SS5, SS6 and SS9.

(4) Specification of the holder task attributes of SS1, SS2 and SS3.

(5) Saving of the SSD object containing the definition of subsystem SS1. In addition, the SSD object is saved to the ISAM file SYSSSC.SS1.001 specified under (1).

4.5.2 Creation of a static subsystem catalog

```
/START-SSCM
//START-CATALOG-CREATION CATALOG-NAME=KAT1 ————————————————————————————  (1)
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSC.SS1.001,
        INSTALLATION-USERID=*UNCHANGED,CORRECTION-STATE=*UNCHANGED ————————  (2)
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSD.SS2.001,
        INSTALLATION-USERID=*UNCHANGED,CORRECTION-STATE=*UNCHANGED ————————  (3)
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=SS3(VERSION=V2.5A00),
        :
        : ————————————————————————————————————————————————————————————————  (4)
//CHECK-CATALOG ————————————————————————————————————————————————————————————  (5)
//SAVE-CATALOG CATALOG-NAME=*CURRENT,FORCED=*NO ————————————————————————————  (6)
```

(1) Declaration of the static subsystem catalog.

(2) Addition of the subsystem definition in the object form for SS1 to the subsystem catalog.

(3) Addition of the subsystem definition in the UGEN format for SS2 to the subsystem catalog.

(4) Definition of the attributes of the SS3 subsystem.

(5) Performance of a catalog check.

(6) Saving of the resultant static subsystem catalog.

4.5.3 Modification of a static subsystem catalog

```
/START-SSCM
//START-CATALOG-MODIFICATION CATALOG-NAME=KAT1 ————————————————————  (1)
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSC.SS5,
        INSTALLATION-USERID=*UNCHANGED,CORRECTION-STATE=*UNCHANGED ———  (2)
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSD.SS4,
        INSTALLATION-USERID=*UNCHANGED,CORRECTION-STATE=*UNCHANGED ———  (3)
//SET-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=SS1(V01.0A10),
        LINK-ENTRY=SS1ENT,MEMORY-CLASS=*SYSTEM-GLOBAL(SUBSYSTEM-ACCESS=
        *SYSTEM),INIT-ROUTINE=INITROUT,DEINIT-ROUTINE=*DYNAMIC,
        INTERFACE-VERSION=INTVERS ————————————————————————————————  (4)
//SEPARATE-ADDRESS-SPACE SUBSYSTEM-NAME=SS1,
        FROM-SUBSYSTEMS=(SS5,SS6) ————————————————————————————————  (5)
//ASSIGN-HOLDER-TASK TYPE=*WORK-TASK(SUBSYSTEM-NAME=SS1,
        SUBSYSTEM-VERSION=V01.0A10) ———————————————————————————————  (6)
//REMOVE-CATALOG-ENTRY SUBSYSTEM-NAME=SS7(VERSION=V03.2A00) ————————  (7)
//MODIFY-SUBSYSTEM-ATTRIBUTES SUBSYSTEM-NAME=SS2(VERSION=V11.0A10),
        INSTALLATION-USERID=UIDXYZ,....
        : ————————————————————————————————————————————————————————  (8)
//REMOVE-ADDR-SPACE-SEPARATION SUBSYSTEM-NAME=SS1,
        FROM-SUBSYSTEMS=SS2 ——————————————————————————————————————  (9)
//MODIFY-WORK-TASK-ATTRIBUTE SUBSYSTEM-NAME=SS3,
        SUBSYSTEM-VERSION=00.1,WORK-TASK=*NO  ————————————————————— (10)
//CHECK-CATALOG ————————————————————————————————————————————————— (11)
//SAVE-CATALOG CATALOG-NAME=*CURRENT,FORCED=*NO ————————————————— (12)
```

(1) Specification of the catalog whose contents are to be modified.

(2) Addition of a subsystem definition contained in an SSD object.

(3) Addition of a subsystem definition that was written in the old DSSMGEN syntax (UGEN format).

(4)-(6) Addition of a new subsystem and specification of its attributes.

(7) Deletion of a subsystem definition from the catalog.

(8) Modification of a subsystem definition which has already been entered in the catalog.

(9) Removal of the strict address space separation on both subsystems.

(10) Modification of the work task attributes of a subsystem which is being used as a work task.

(11) Performance of a catalog check.

(12) Saving of the modified static subsystem catalog.

5 Related publications

You will find the manuals on the internet at https://bs2manuals.ts.fujitsu.com .

ADAM (BS2000)
Abstract Device Access Method
User Guide

AID (BS2000)
Advanced Interactive Debugger
Core Manual
User Guide

ARCHIVE (BS2000)
User Guide

BLSSERV
Dynamic Binder Loader / Starter
User Guide

BINDER (BS2000)
User Guide

CALENDAR (BS2000)
User Guide

CRTE (BS2000)
Common RunTime Environment
User Guide

DAB (BS2000)
Disk Access Buffer
User Guide

BS2000 OSD/BC
Utility Routines
User Guide

Distributed Print Services (BS2000)
Printing in Computer Networks
User Guide

DRV (BS2000)
Dual Recording by Volume
User Guide

EDT (BS2000)
Statements
User Guide

BS2000 OSD/BC
Introductory Guide to DMS
User Guide

BS2000 OSD/BC
Introduction to System Administration
User Guide

BS2000 OSD/BC
System Installation
User Guide

FDDRL (BS2000)
User Guide

HSMS (BS2000)
Hierarchical Storage Management System
2 Volumes
User Guide

Volume 1 contains the description of the functions, management and the installation
Volume 2 contains the description of the HSMS statements in alphabetical order

IMON (BS2000)
Installation Monitor
User Guide

JV (BS2000)
Job Variables
User Guide

BS2000 OSD/BC
Commands (Volume 1-7)
User Guide

LMS (BS2000)
SDF Format
User Guide

MAREN (BS2000)
Volume 1 and 2
User Guides

Volume 1 contains an introduction to working with MAREN.
Volume 2 This covers overviews, interface descriptions and examples of working with MAREN, divided into a privileged and a nonprivileged section.

HIPLEX MSCF (BS2000)
BS2000 Processor Networks
User Guide

PCS (BS2000)
Performance Control Subsystem
User Guide

POSIX (BS2000)
POSIX Basics for Users and System Administrators
User Guide

POSIX (BS2000)
Commands
User Guide

PROP-XT (BS2000)
Programmed Operating with SDF-P
Product Manual

RFA (BS2000)
Remote File Access
User Guide

RSO (BS2000)
Remote SPOOL Output
User Guide

SDF-A (BS2000)
User Guide

SDF-P (BS2000)
Programming in the Command Language
User Guide

SDF (BS2000)
SDF Management
User Guide

SECOS (BS2000)
Security Control System
User Guide

openSM2 (BS2000)
Software Monitor
User Guide

SPOOL (BS2000)
User Guide