EUROMAP 63

Data Exchange Interface

Version 1.05a
July, 2000
(59 pages)

The recommendation under this cover has been prepared by the Technical Commission of EUROMAP and SPI (The Society of the Plastics Industry, USA-Washington, DC)

In this document, the American spelling is used.

In the U.S. an identical text of EUROMAP 63 is published as an SPI document.

Table of Contents

1. INTRODUCTION 4
1.1. EUROMAP - SPI COMMITTEE ON COMMUNICATION PROTOCOL 4
1.2. FILE BASED COMMUNICATION (FBC) 5
1.3. MANUAL ORGANIZATION 6
2. CCP PROTOCOL DEFINITION 6
2.1. OSI SEVEN LAYER MODEL 6
2.2. LAYER 1, PHYSICAL LAYER 7
2.2.1. CCP File Based Communicaton (FBC) Physical Standard 7
2.3. LAYER 2, DATA LINK LAYER 7
2.3.1. CCP FBC Data Link Standard 7
2.4. LAYER 3, FBC NETWORK LAYER 7
2.4.1. CCP FBC Network Standard 7
2.5. LAYER 4, FBC TRANSPORT LAYER 7
2.5.1. CCP FBC Transport Standard 9
2.6. LAYER 5, SESSION LAYER 9
2.6.1. CCP FBC Communication Sessions 9
2.6.2. Session Request and Response File (stream) Formats 10
2.6.2.1. Session Request File Format 10
2.6.2.2. Session Response File Format 12
2.6.2.3. Session Request File Commands and Responses 12
2.7. LAYER 6, PRESENTATION LAYER 13
2.7.1. CCP FBC Presentation Layer 13
2.7.1.1. Physical Data Formats 13
2.7.1.2. Presentation Request and Response Command Files 14
2.7.1.2.1. Presentation Request and Response File Formats 14
2.7.1.2.2. Presentation Request File Format 14
2.7.1.2.3. Presentation Response File Format 15
2.7.1.2.4. Presentation Requests and Responses 16
2.8. LAYER 7, APPLICATION LAYER 16
2.8.1. Device specific command tokens and formats 18
2.8.2. File formats 20
2.8.2.1. Report Response File Format 20
2.8.2.2. Event Log Response File Format 21
2.8.2.2.1. Machine Alarms Event Protocol File Format 22
2.8.2.2.2. Current Machine Alarms Event Protocol File Format 22
2.8.2.2.3. Overall Change Event Protocol File Format 22
2.8.2.3. GETINFO Response File Format 23
2.8.2.4. GETID Response File Format 23
2.8.2.5. General Information File Format 24
2.8.3. Event Log Types and Token 25
3. APPENDICES 25
3.1. ABBREVIATIONS AND ACRONYMS 25
3.2. WHITE SPACES AND SPECIAL CHARACTERS 26
3.3. COMMUNICATION SESSION FLOW 27
3.3.1. Session Request Flow Chart 27
3.3.2. Session Replay Flow Chart 28
3.4. COMMUNICATION PRESENTATION FLOW 29
3.5. ERROR CLASS CODES 30
3.6. SESSION LAYER ERROR CODES 30
3.7. SESSION REQUEST COMMANDS AND RESPONSES 31
3.7.1. Session Connection Verification Command 31
3.7.2. Session Execute Command 31
3.8. PRESENTATION LAYER ERROR CODES 33

3.9. PRESENTATION REQUEST AND RESPONSE PARAMETERS 34
3.9.1. File Specification Parameter 34
3.9.2. Time Specification Parameter 34
3.9.3. Date Specification Parameter 35
3.9.4. Comparison Operator Specification 35
3.9.5. Constant Parameter Specification 35
3.9.6. Application Parameter Specification 36
3.9.7. Start Specification Parameter 36
3.9.8. Stop Specification Parameter 37
3.10. PRESENTATION REQUEST COMMANDS AND RESPONSES 38
3.10.1. Presentation Job Request Command 38
3.10.2. Report Generation Request Command 38
3.10.3. Event Log Request Command 43
3.10.4. Abort Request Command 44
3.10.5. Upload Request Command 44
3.10.6. Download request Command 45
3.10.7. GETINFO request Command 46
3.10.8. GETID request Command 48
3.10.9. SET request Command 48
3.11. IMM DEVICE TOKEN SUMMARY 50
3.12. TOKEN SUMMARY FOR OTHER DEVICES 54
3.13. UNIT SUMMARY 54
3.14. APPLICATION EXAMPLES 55
3.14.1. Recording of Process Log 55
3.14.2. Recording of Alarm Log 56
3.14.3. Recording of Process Log for a Status View 56
3.14.4. Recording of Alarm Log for a Status View 57
3.14.5. Download a Data Set (Example 1) 57
3.14.6. Download a Data Set (Example 2) 58
3.14.7. Setting Several Parameters 58
3.14.8. GETID Command 59

List of Tables and Figures

Figure , File Based Communication 5
Figure , OSI Seven Layer Model 6
Table , FBC Physical Data Formats 13
Table , Presentation Layer White-Space characters 26
Table , Type Token Group 18
Table , Unit Token Group 19
Table , Function Token Group 19
Table , Miscellaneous Token Group 20
Figure , Session Request Flow Chart 27
Figure , Session Replay Flow Chart 28
Table , Error Class Codes 30
Table , Session Layer Error Codes 30
Table , Presentation Layer Error Codes 33
Table , Machine Status Tokens 51
Table , Barrel Temperature Tokens 52
Table , Mould Temperature Tokens 52
Table , Miscellaneous Temperature Tokens 53
Table , Process Monitoring Parameter Tokens 54
Table , Physical Units Used in Injection Moulding Machines 55

1. Introduction

1.1. EUROMAP - SPI Committee on Communication Protocol

The widespread use of microprocessor controls in industrial equipment allows data to be collected and transmitted to central computer systems. Since the mid-eighties, there have been efforts, largely driven by the automotive industry, to establish computer integrated production plants. The aim of these efforts is to increase the transparency of the production processes and thus to obtain information rapidly concerning throughput, quality and utilization.

The benefits of networked systems are also important to the plastics industry; thus in 1992 the EUROMAP 15 standard for injection molding machine communication was established. Likewise, in 1994, the SPI organization developed standards to encompass similar communication systems in the SPI Phase II Communication Protocol.

Nevertheless, it has not been possible to establish either protocol widely. Because of the rapid developments in computer technology, certain transfer techniques quickly became obsolete and unacceptably expensive. In addition, the compatibility between the various manufacturer’s products was many times uncertain. Thus, the implementation and servicing of the existing protocols has led to much expense on the part of machinery manufactures as well as end processors.

The joint EUROMAP - SPI Committee on Communication Protocol (CCP) was assigned the task to standardize a protocol for use by plastic machinery manufacturers. The committee was charged to develop a flexible concept, by use of simple communication structures, which establishes an economical host computer interface using standard data processing hardware and software. The committee adopted the following rules to govern the creation of this protocol:

- ***The protocol will be developed using as many existing standards and protocols as possible.***
  This rule was designed to help save in development time and add credibility to the final protocol. Also, it will aid in developer and end user understanding as there are existing books and reference documents which describe the various standards and protocols.

- ***The protocol shall use existing and readily available technology and shall not interfere with the normal operations of the processing equipment.***.
  As many of the manufacturers had already developed various equipment with communication capabilities. the protocol had to be able to be retrofitted in existing equipment designs. The addition of the communication hardware and software must not impose any impact on the main operating functionality of the processing equipment.

- ***The purpose of the protocol is to provide centralized setup and monitoring of various plastics processing equipment from a centralized computer system. The protocol shall not be used for any function necessary for the safe operation of equipment nor perform any direct control of equipment.***
  The central computer would not actually control the equipment, but would allow the central computer to set a machine's operating parameters and then monitor it's process data. No direct control functions or function necessary for the safe operation of equipment will be provided by the protocol. At no time will any control or safety function of equipment be dependent on the operation of the communication network.

1.2. File Based Communication (FBC)

In order to satisfy the general requirements adopted by the committee, a File Based Communication (FCB) model has been created. This model can be described using the following structure:

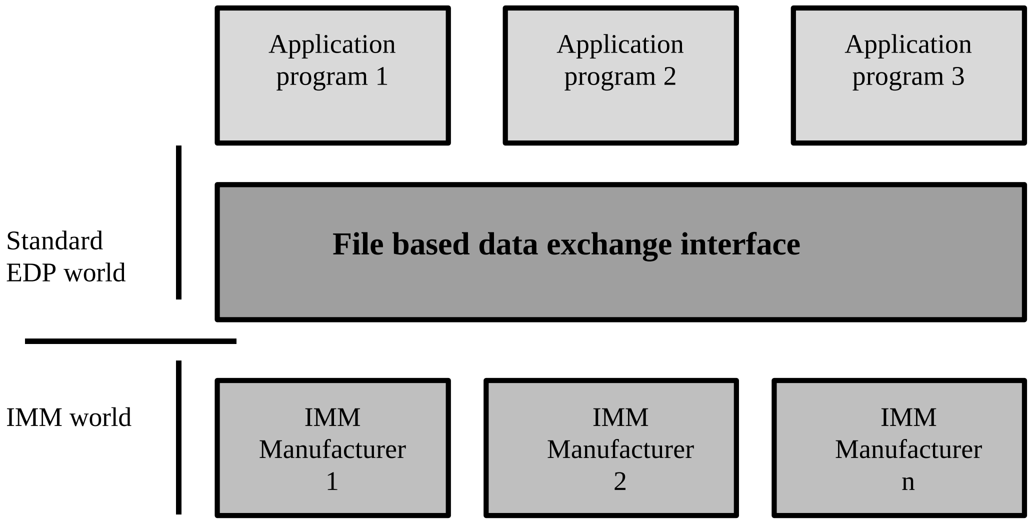

Figure , File Based Communication

As can be seen from the schematic representation, the file based data exchange interface provides a standard method for different application programs to access information from various machines regardless of manufacturer or type of control systems utilized.

The interface itself is accomplished using ASCII files. An underlying network file system, whose type and mechanisms are outside the scope of this document, is responsible for providing access to shared network file servers. Files termed “Job Files” are created to identify the information to be communicated with a particular machine. The machine will process this job file to both receive information as well as identify information to be written by the machine to a “Response File”. Within these files, commands and response are written using a simple syntax based on ASCII command tokens. For example, a job file for sending a complete set of machine setup parameters to a machine might have the following appearance (keywords in capital letters):

```
JOB test RESPONSE = test.log; // Specification of the data set
UPLOAD “\\sv1\vol1\data\test”// Use data from data set „test“
START TIME>= 10:05:00;        // Start upload at 10:05
```

This method allows the utilization of sophisticated and complex software applications to access machine information by automatically creating job files and processing returned response files as well as the use of a simple text editor to manually create job files and view returned response files. Further, this method allows the creation of a complete process monitoring system using standard data processing (personal computer) components. For example, a word processing program can be used to created job request files and a spreadsheet program can be used to analyze the returned machine information. In addition, existing process monitoring systems can be easily adapted to use this file based communication method in place of costly, slow and unreliable direct serial connections.

1.3. Manual Organization

The CCP Version 1.00 manual was organized based on the Open Systems Interconnection (OSI) 7 Layer Model. This model provides a basis for developing and discussing the hierarchical layers of communication protocols. The model provides for different standards and protocols for each layer allowing the CCP Protocol to be enhanced in the future to take advantage of newer technologies while maintaining support of existing procedures.

The **CCP Protocol Definition** section describes the standards, procedures, and protocols defined or required for each of the seven OSI model layers. Each description provides a reference to the existing standards on which the layer is based along with a description of it's procedures. It is intended that this document serve as an overview of the various specifications and standards, however, some of the referenced documents may be required for protocol understanding and implementation. Where applicable, multiple (mutually exclusive) standards are presented which can be used in different CCP Protocol applications.

The document is concluded with various appendices which provide additional information concerning the protocol.

2. CCP Protocol Definition

2.1. OSI Seven Layer Model

The OSI Seven Layer Model is used to describe and design communication protocols. The model was developed in 1984 by the International Organization for Standardization. The model breaks the overall communication process into 7 hierarchical layers. Standards, procedures, and protocols are then defined for each layer. The compilation of standards for each layer forms an overall computer protocol application. The OSI model was chosen to describe the CCP Protocol as it has become the most common framework used to discuss communication protocols.

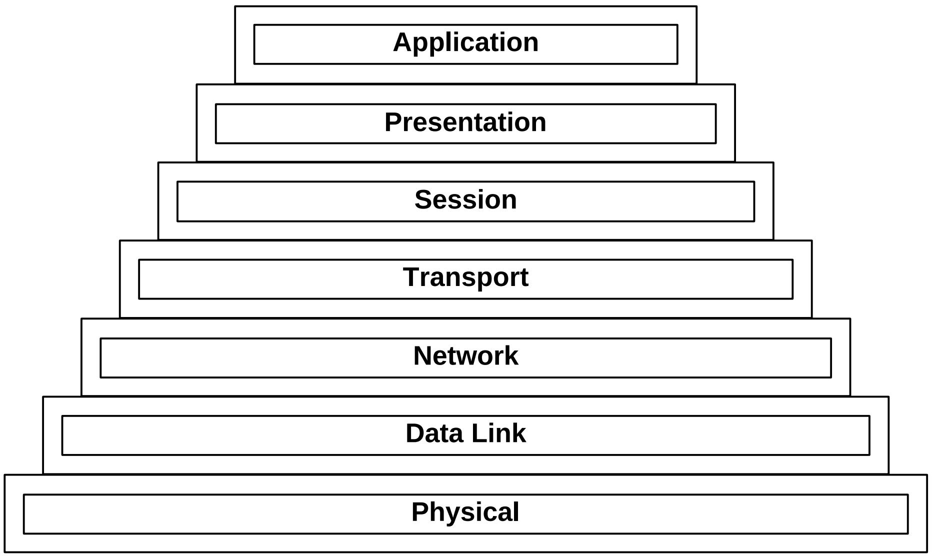

Figure , OSI Seven Layer Model

The joint EUROMAP - SPI CCP documentation defines only the top three layers of the communication model. The lower four layers are left for definition by actual implementations of the protocol. An underlying network system which provides Network / Remote File System services is expected to exist at these layers. These services are required to locate and open files, detect file changes, lock files from shared access, read and write data to files and close files. Guidelines and recommendations for the implementation of these lower layers are offered.

2.2. Layer 1, Physical Layer

Layer 1 of the OSI model is the Physical Layer. This layer consists of the electrical and mechanical specifications of the communication link. The layer defines mechanical interface specifications, electrical bit representation specification, bit stream procedures, and transmission activation, maintenance, and deactivation procedures. The collection of these procedures forms the Physical Layer.

2.2.1. CCP File Based Communicaton (FBC) Physical Standard

This layer is not specifically defined by this protocol. However this protocol does require the existence of this layer in any actual implementation. Examples of specifications at this layer suitable to support the CCP upper layers are twisted pair and 10BaseT as defined by the IEEE 802.3 (Ethernet) protocol.

2.3. Layer 2, Data Link Layer

The OSI Data Link Layer defines the methods and procedures used to establish, maintain, and release data link connections. It also specifies the network topology, line discipline, flow control, and error control.

2.3.1. CCP FBC Data Link Standard

This layer is not specifically defined by this protocol. However this protocol does require the existence of this layer in any actual implementation. It is assumed that the underlying network system in place used to implement the network / remote file system provides these services. An example of a standard which satisfies the requirements at this layer is CSMA/CD.

2.4. Layer 3, FBC Network Layer

The OSI Model Network Layer is responsible for establishing, maintaining and terminating network connections. The layer provides the mechanisms to communicate with connected networks and sub-networks.

2.4.1. CCP FBC Network Standard

This layer is not specifically defined by this protocol. However this protocol does require the existence of this layer in any actual implementation. It is assumed that the underlying network system in place used to implement the network / remote file system provides these services. An example of a standard which satisfies the requirements at this layer is TCP/IP.

2.5. Layer 4, FBC Transport Layer

The OSI Model Transport Layer is responsible for transferring data between the upper and lower layers of the OSI model. The layer prevents the upper layers from requiring details on how data is communicated by the lower layers.

2.5.1. CCP FBC Transport Standard

This layer is not specifically defined by this protocol. However this protocol does require the existence of this layer in any actual implementation. It is assumed that the underlying network system in place used to implement the network / remote file system provides these services. An example of a standard which satisfies the requirements at this layer is TCP/IP.

2.6. Layer 5, Session Layer

The OSI Model Session Layer provides a variety of services and functions which are used by the application layer to access the communication network. Such services include the establishment of session connections, data exchange, data synchronization, semaphore processing, and other functions.

2.6.1. CCP FBC Communication Sessions

The EUROMAP **-** SPI Data Exchange Interface relies on the Session Layer to initiate communication sessions. Communication sessions are initiated by placing a Session Request file in a particular machine's Session Directory. The location of the Session Directory is specified in the Machine Initialization file. Each machine will define the maximum number of communication sessions that can be activated at a time (MaxSessions). This value is documented in the Machine Initiation file as well as returned as a response to the GetInfo application level request. The MaxSessions value is used to form the Session Request File name as follows:

SESSnnnn.REQ

Where

SESS – the ASCII characters "SESS"
nnnn – an open Session Number. The number is a four character ASCII numeric text string and is valid from '0000' MaxSessions-1. For example, if a machine reports MaxSessions as 4, SESS0000.REQ, SESS0001.REQ, SESS0002.REQ and SESS0003.REQ are the only valid Session Request File names. An application must first search the Session Directory for an available / open Session Number. If all valid session numbers are already taken (al valid Session Request files exist in the directory), an error should be reported. Otherwise, the application should use the first available Session Number in the formation of the Session Request file name.
.REQ – the file extension .REQ to indicate it is the Session Request file.

Once a Session Request file is created in the Session Directory, the application relinquishes control of the file. The responder (machine), upon detection of the new Session Request file, will open the file and begin to process the Session Request Commands. As the commands are processed, session layer response information is written to a Session Response file. This file is created using the same base name as the Session Request file but with an extension of ".RSP". The file is created in the same directory location as the Session Request file.

Session Request Files contain Session Request Commands. These commands are used to initiate various communication procedures and functions. The commands consist of Session Request Command Tokens and appropriate parameters. The responder will execute each Session Request Command in order, writing acknowledgement information to the Session Response File. If a session request command is a valid command and is executed by the responder, a positive acknowledgment is written to the session response file. If, for any reason, a session request command cannot be executed by the responder, an appropriate negative (error) acknowledgment is written to the session response file.

Once all Session Request commands have been processed by the responder, the Session Request File is deleted by the responder and the Session Response File is closed. The original requestor, upon detection of the deleted Session Request File and closed Session Response File, can review the response file for session command execution conformation. When complete, the requestor deletes the Session Response File and the communication session is considered closed. The Session Number used by the completed communication session is once again open and available for use.

Note that session communication requests are processed immediately (within the session request response time) by the responder and are then terminated. Session requests do not remain active over time. However, presentation and application layer requests may remain active after the initiating session layer request is processed and terminated. It is the responsibility of those layers to provide the appropriate synchronization and termination services if such capabilities are provided.

Appendix 3.3, Communication Session Flow, provides a flow chart of the communication session process.

2.6.2. Session Request and Response File (stream) Formats

Both session request and response files are ASCII based text files. The files consist of command tokens delimited by ***white-space***.

Definitions see at appendix “White Spaces and Special Characters”.

2.6.2.1. Session Request File Format

All session request files are ASCII based text files. Commands are stored as single language tokens. The basic command format is as follows:

{Session Command Identifier} {Request Command} {Parameters} ;

where

- All keywords are written in capital letters and the interpretation is casesensitive
- Session Command Identifier **-** a unique 8 character code which is used to identify the communication session request. The code must be unique within this session request file. The Session Identifier is terminated by white space. Typically, this code is set as an 8 character ASCII numeric text string starting with “00000000”, “00000001”, etc.
- Request Command **-** the session request command token of the command to be processed. The Request Command is terminated by white space.
- Parameters **-** session command dependent parameter tokens
- ; **-** the end of session command token

The Session Command Identifier token is used to uniquely identify a communication session command. The token is created by the requester and is used when a session request command is issued. The responder uses this token when writing the request acknowledgment information in the session response file.

2.6.2.2. Session Response File Format

All session response files are ASCII based text files. Responses are stored in one of the following formats:

{Session Command Identifier} PROCESSED {Response Information} ;
{Session Command Identifier} ERROR {class}
{nnnnnnnn} {Error Description} ;

where

- Session Command Identifier - the unique 8 character code which was specified by the requester and is used to identify the communication session request being responded to. The Session Identifier is terminated by a white space.
  If the Command Identifier is not valid, in the response file will be written “????????” instead of the Command Identifier.
- PROCESSED - the keyword PROCESSED indicating that the request was processed by the responding station (a positive acknowledgment). Following this keyword is any request command specific information. The PROCESSED keyword is terminated by a white space.
- ERROR - the keyword ERROR indicating that the request was not processed by the responding station (a negative acknowledgment). The ERROR response contains the following additional information:

  class - the 2 character error class code. Errors detected and generated at the session layer are assigned the class 05 (represented in ASCII format). If the session layer detects an error from a lower layer service it will report that service’s error class and information if possible. Errors detected and generated at the presentation and application layers must provide the appropriate class code and error information. The class is terminated by a white space.

  nnnnnnnn - the session response error code represented as a numeric value in ASCII format with leading zeroes to fill to 8 characters. Appendix 3.3, Session Request Commands and Responses, provides a detailed description of all session response error codes. The error code is terminated by a white space.

  Error Description - max. length 255 character string in quotes, padded with spaces (SPACE) if necessary, which describes the error condition. Any white-space characters within the string are considered as part of the string.

- ; - the end of session command token

The Session Command Identifier token of the acknowledgment command must be that which was originally specified by the requester in the request command.

2.6.2.3. Session Request File Commands and Responses

Session Request File Commands are used by the requesting network station to initiate a communication session. The commands are initiated by the presentation and application layers via session layer service calls. These service calls provide the basic operations of connection verification, configuration verification, and presentation and application layer specific command execution. Each command also specifies appropriate response information, if any, to be provided by the responder. Appendix 3.3, Session Request Commands and Responses, provides a complete list of all session request commands, responses, and formats.

2.7. Layer 6, Presentation Layer

The OSI Model Presentation Layer is concerned with the transfer syntax used between communicating systems. The layer requires that communicating devices determine and use a compatible transfer syntax to exchange application data.

2.7.1. CCP FBC Presentation Layer

This CCP Presentation Layer is based on a File Based Communication (FBC) network architecture. The layer is dependent on the CCP FBC Session Layer services but is otherwise independent of lower layer functionality. The layer is responsible for defining the basic data formats and overall data representation syntax.

2.7.1.1. Physical Data Formats

The FBC presentation layer specifies various data formats for storing information. Different data formats are used according to the requirements of the specific commands. The following general rules apply to all specified data formats:

- All multi-byte data is represented in ASCII
- Fixed length character strings are padded with SPACE characters to fill the entire data field.

The following table defines the various FBC data formats.

Data Type	**Byte Length**	**Numeric Range**	**Description**
CHAR	1	n/a	Single character data. (Unicode not supported).
FSTRING(n)	n	n/a	Fixed length character string, space (SPACE) padded to (n) characters.
VSTRING(n)	n	n/a	Variable length string, <= n characters. The string is delimited with the quote characters (“”). A quote character within a string is represented with two quote characters in sequence (““)
NUMERIC(n)	n, where n <= 16	n/a	Representation in the following standard numeric formats: [-]mmm (integer value) [-]mmm.ddd (float value) [-]m.ddd**e**[±]xx, [-]m.ddd**E**[±]xx (float value in scientific format)

Table , FBC Physical Data Formats

2.7.1.2. Presentation Request and Response Command Files

The CCP Presentation Layer provides ***command files*** for execution of application and presentation layer functions. The application layer combines presentation layer commands and data formats with application layer specific data values to form a command file for execution by a network station. The command file is stored on a network storage device accessible by the remote network station (machine) and then is transferred to the network station via the session layer EXECUTE service.

The responding station, upon detection of the SESSION level EXECUTE request, passes the EXECUTE information to its presentation layer for processing. The responder's presentation layer processes the command file, passing all application layer information to that layer. Once processing of the presentation and application layer information is complete, the responding presentation layer returns an appropriate positive or negative acknowledgment to the underlying session layer to complete the communication request.

All presentation layer command files specify a corresponding response file for various response information to be stored by the responding presentation layer. The response file will contain response information for each command which is processed by the presentation and application layers.

2.7.1.2.1. Presentation Request and Response File Formats

Both presentation request and response files are ASCII based text files. The files consist of command tokens delimited by ***white-space***. The following characters are considered white-space:

Definitions see at appendix "White Spaces and Special Characters".

2.7.1.2.2. Presentation Request File Format

All presentation request files are ASCII based text files. Commands are stored as single language tokens. The basic command format is as follows:

- {Request Command} {Parameters} ;

where

- All keywords are written in capital letters and the interpretation is casesensitive
- Request Command - the presentation request command token of the command to be processed. The request command is terminated by white space.
- Parameters - presentation command dependent parameter tokens. The parameters are separated by white space. Note: If a parameter (IMM Token) is not supported by the

machine, the syntax parser throws the error 00000006 Unknown REPORT parameter.

- ; **-** the end of command token

The responding presentation layer processes the command file command by command. The first command of every command file is the JOB command. This command provides information to the responder required to execute the request file. Included in this command is the file specification of the command response file. This file is used to store response information to each session request command and is created by the responder (with any existing file information being destroyed) when the JOB command is executed. As the request commands are processed, appropriate response information is written to the response file for each command. Once all request file commands have been processed, the response file is closed and a positive acknowledgment is returned to the session layer. The request is now complete at the presentation layer.

A negative acknowledgment is returned to the session layer if one or more of the following conditions are true:

1. The responder is unable to locate, open, or read from the request command file as specified by the session layer EXECUTE command.
2. The request command file cannot be executed at this time (for any reason).
3. The JOB command was not the first request file command or the command syntax was invalid.
4. The response file cannot be created or written to by the responder.

Otherwise, a positive acknowledgment is returned to the session layer. Note that a positive acknowledgment only indicates that the file specified by the session layer EXECUTE request was processed and a corresponding presentation layer response file was created. It does not indicate that the responder was able to process all presentation or application layer information within the presentation layer request file. All presentation and application specific command response information is stored within the presentation layer response file.

2.7.1.2.3. Presentation Response File Format

All presentation response files are ASCII based text files. Responses are stored as single language tokens. The basic response format is one of the following:

```
COMMAND {n} PROCESSED {Command Response}
                {date_spec} {time_spec};
COMMAND {n} ERROR {class}
                {xxxxxxxx} {Error
          Description}
                {date_spec} {time_spec};
```

where

- COMMAND - the keyword COMMAND which indicates the start of a command response. The COMMAND keyword is terminated by white space..
- n - the request file command number. The commands are sequentially numbered, starting with 1, as they are executed. The command number is used to synchronize requests and responses. The command number is terminated by white space.
- PROCESSED - the keyword PROCESSED if the command was processed by the responder. Following this keyword is any command specific response data. The PROCESSED keyword is terminated by white space.
- ERROR - the keyword ERROR if the command was not processed by the responder. The ERROR response contains the following additional information (with each being terminated by white space):
    - class - the 2 character error class code. Errors detected and generated at the presentation layer are assigned the class 06 (represented in ASCII format). Errors detected and generated at the application layer must provide the appropriate class code.
    - xxxxxxxx - the presentation response error code. Appendix, Presentation Request and Response Commands, provides a detailed description of all presentation response error codes.
    - Error Description - a VSTRING(255) string which describes the error condition in quotes.
- date_spec - date of occurrence (separated by white space).
- time_spec - time of occurrence (separated by white space).
- ; - the end of response token

2.7.1.2.4. Presentation Requests and Responses

Presentation Requests are issued to perform various presentation layer services using information supplied by the application layer. These services include reading application layer information, writing application layer information, and executing application layer functions. The presentation layer handles the common command execution processing while the application layer provides and processes the actual information.

Each presentation layer request defines its syntax according to its specific requirements. Where required, application layer data is identified. The format of the expected response information is also specified by each presentation request command. Appendix 3.8, Presentation Request and Response Commands, provides a detailed description of all request commands and responses.

2.8. Layer 7, Application Layer

The OSI Model Application Layer is the topmost layer of the model. The main function of this layer is to allow the exchange of information by distributed application processes. It is at this layer that all underlying layer usage requirements are defined to form a complete protocol.

2.8.1. Device specific command tokens and formats

The following conventions have been developed to help with the naming of command data tokens. These conventions should be maintained whenever possible.

Command data tokens are named according to their type, units, function and miscellaneous additional information. Charts of common abbreviations have been created for each of the categories named above. The first letter of each abbreviation is in upper case while all remaining letters are in lower case. An abbreviation from each category is concatenated to form the full token name. For example, the token for the maximum machine hydraulic pressure measured during a cycle would be represented by the following abbreviations:

Act **-** for an Actual value (Type Group)
Prs **-** for a Pressure value in bar (Units Group)
Mach **-** for a Machine (or system) value (Function Group)
Hyd **-** for a hydraulic pressure value (Miscellaneous Group)
Max **-** for a maximum measurement over a period of time (Miscellaneous Group)

The abbreviations are then concatenated to form ***ActPrsMachHydMax***.

When creating vendor specific tokens outside of the tokens defined by the committee, the conventions described here should be followed. To identify the token as a vendor specific token, the at symbol '@' is used as a prefix to the token symbol.

Wildcards at parameter arrays in JOB files will **not be allowed** (i.e.ActTmpBrlZn[*,*]).

Token Abbreviation	**Full Token**	**Description**
Set	**Set**	Specifies a value set at the machine.
Act	**Act**	Specifies an actual value measured at the machine.

Table , Type Token Group

Token Abbreviation	**Full Token**	**Units**	**Description**
Cfg	**C**on**f**i**g**uration	n/a	Used to specify the configuration or setup of a function.
Vol	**Vol**ume	ccm	Used to specify a volumetric value.
Vel	**Vel**ocity	mm/s	Used to specify a velocity value.
Tim	**Tim**e	s	Used to specify a time measurement.
Tmp	**T**e**mp**erature	C	Used to specify a temperature measurement.
Spd	**Sp**ee**d**	1/min	Used to specify a speed measurement.
Prs	**Pr**e**s**sure	bar	Used to specify a pressure measurement.
Pwr	**P**o**w**e**r**	W	Used to specify a power measurement.
Str	**Str**oke	mm	Used to specify a stroke or linear position measurement.
Cnt	**C**ou**nt**	n/a	Used to specify a count.
Fce	**F**or**ce**	kN	Used to specify a force measurement.
Dia	**Dia**meter	mm	Used to specify the diameter.

Token Abbreviation	**Full Token**	**Units**	**Description**
Desc	**Desc**ription	n/a	Used to describe something.
Sts	**St**atu**s**	n/a	Used to specify the active status of a function.
Rec	**Rec**ipe	n/a	Used to specify the recipe or setup of a function.

Table , Unit Token Group

Token Abbreviation	**Full Token**	**Description**
Inj	**Inj**ection	Injection function.
Eje	**Eje**ctor	Ejector function.
Mld	**M**o**ld**	Mold function.
Clp	**Cl**am**p**	Clamping unit function.
Brl	**B**a**r**re**l**	Barrel function.
Plst	**Pl**a**st**icise	Plasticise or Recovery function.
Scr	**Scr**ew	Screw function.
Xfr	Transfer	Transfer, Switch Over, or Cut-off function.
Cav	**Cav**ity	Cavity function.
Hld	**H**o**ld**	Hold function.
Csh	**C**u**sh**ion	Cushion function.
Dcmp	**D**e**c**o**mp**ress	Decompress function.
Mach	**Mach**ine	Machine or System function.
Cyc	**Cyc**le	Cycle function.
Prt	**P**a**rt**	Part function.
Job	**Job**	Overall job or process function.
Op	**Op**erator	Operator function.
Mat	**Mat**erial	Material function.
Oil	**Oil**	Oil function.
Wtr	**W**a**t**e**r**	Water function.
Cab	**Cab**inet	Control cabinet function.
Mlt	**M**e**lt**	Melt function.

Table , Function Token Group

Token Abbreviation	**Full Token**	**Description**
Hdev	**H**igh **dev**iation	High deviation limit - based relative to another token value.
Ldev	**L**ow **Dev**iation	Low deviation limit - based relative to another token value.
Hlmt	**H**igh **l**i**m**i**t**	High absolute limit.
Llmt	**L**ow **l**i**m**i**t**	Low absolute limit.
Pre	**Pre**	Used to specify the before state for functions which can occur both before and after another function (such as decompression).
Pst	**P**o**st**	Used to specify the after state for functions that can occur both before and after another function (such as decompression).
Max	**Max**imum	Used to specify the maximum value over a range of values or time.
Min	**Min**imum	Used to specify the minimum value over a range of values or time.
Ave	**Ave**rage	Used to specify the average of values over a range or time.
Spec	**Spec**ific	Used to represent specific pressure.
Hyd	**Hyd**raulic	Used to represent hydraulic pressure.
In	**In**	Used to represent the input portion of a function.
Out	**Out**	Used to represent the output portion of a function.
Rej	**Rej**ect	Used to specify a rejected function.
Lot	**Lot**	Used to represent a particular lot or group.
Nxt	**N**e**xt**	Used to represent the next item in a group.
Box	**Box**	Used to represent a single collection or group.
Zn	**Z**o**n**e	Used to represent a zone of a function.
Stb	**St**and**b**y	Used to represent the standby or alternate condition of a function.

Table , Unit Miscellaneous Group

2.8.2. File formats

2.8.2.1. Report Response File Format

Report files are created in response to job file REPORT commands. The REPORT command is used to generate an application data report and is described in Section 3.8. The requested application data is written to Report Response Files.

All report response files are ASCII based text files. The files consist of values delimited by the list delimiter LD (see GETINFO request command) and terminated by the EOL sequence.

The first line of the response file contains the list of reported tokens:

{param_id_1} LD {param_id_2} LD{param_id_3} LD ... {param_id_n} EOL where

{param_id_x} - token which identifies the application layer parameter (type APPLICATION_PARAMETER).

All following lines contain the values of the reported parameters:

{{value_id_1} LD {value_id_2} LD ... {value_id_1} EOL }...

where

value_id_x - value of reported parameter, ASCII numeric or ASCII text in quotes with a maximum of 255 characters (dependent on parameter type).

2.8.2.2. Event Log Response File Format

Event Log Response files are created in response to job file EVENT commands. The EVENT commands are used to generate reports consisting of asynchronous event type data such as alarms or setpoint changes (see section 3.9.3, Event Log Request Command). The requested EVENT data is written to the Event Log Response files.

All Event Log Response files are ASCII based text files. The files consist of values delimited by the list delimiter LD (see GETINFO request command) and terminated by the EOL sequence.

All responses to Event Log request commands share the same base format as follows:

{n} LD {date} LD {time} LD {cycle counter}

where

- n - the unique number identifying the specific event (numeric value in ASCII - max. UINT16). The events are sequentially numbered, starting with 1.
- date - date of event (type DATE_SPEC).
- time - time of event (type TIME_SPEC).
- cycle counter - value of the cycle counter during event (numeric value in ASCII, type NUM_CONST).

This base event response format is identified as {base EVENT response} in subsequent event response file format descriptions.

Thus, the basic format of all EVENT command responses is as follows:

{base EVENT response} LD {additional information} EOL} ...

where

- base EVENT response - the base event response information as described above
- additional information - this information depends on event type and may consist of one or more parameters.

2.8.2.2.1. Machine Alarms Event Protocol File Format

The ALARMS event type is used to log alarm events. The following is the format of the EVENT ALARMS response:

{{base EVENT response} LD {alarm_set} LD {alarm_number} LD {alarm_text} EOL} ...

where

- alarm_set **-** a single character ASCII numeric value where 0 indicates the alarm is reset or clear and 1 indicates that the alarm event is set or active.
- alarm_number **-** a ASCII numeric value indicating the vendor specific alarm number (type NUM_CONST).
- alarm_text **-** a description of the alarm, ASCII text in quotes, max. 255 characters.

2.8.2.2.2. Current Machine Alarms Event Protocol File Format

The CURRENT_ALARMS event type is used to obtain a list of all currently set or active alarms. The format of the EVENT CURRENT_ALARMS is the as specified for the EVENT ALARMS response.

2.8.2.2.3. Overall Change Event Protocol File Format

The CHANGES event type is used to log machine setup parameter changes as entered by the machine operator.

For numeric parameter changes, the format of the EVENT CHANGES response is as follows:

{{basic response format} LD {param_id} LD {old_val} LD {new_val} LD {user_name} LD {user_id} LD {reason} EOL} ...

where

- {param_id} **-** token which identifies the application layer parameter (type APPLICATION_PARAMETER)
- old_val **-** old value of changed parameter, ASCII numeric or ASCII text in quotes max. 255 characters (dependent on parameter type).
- new_val **-** new value of changed parameter, ASCII numeric or ASCII text in quotes max. 255 characters (dependent on parameter type).
- user_name **-** the name of the active operator or user at the time the parameter was changed, ASCII text in quotes max. 255 characters.
- user_id **-** the identification number of the active operator or user at the time the parameter was changed, ASCII numeric.
- reason **-** the reason of change, ASCII text in quotes max. 255 characters.

For other types of EVENT CHANGES parameter changes, the format of the EVENT CHANGES response is as follows:

{basic response format} LD {text} EOL

where

- text - a description of the changes parameter, ASCII text in quotes max. 255 characters.

For example, if an entire parameter set is loaded, the additional information might be set to "DOWNLOAD FROM HOST".

2.8.2.3. GETINFO Response File Format

This response file type is used to contain the results of the GETINFO request command. This command is used to obtain static information concerning a specific machine. The file consists of ASCII text strings delimited by the list delimiter LD (see GETINFO request command) and terminated by the EOC (;) sequence.

The format of a GETINFO response entry is as follows:

{{item_id} LD {item_entry} ;} ...

where

- item_id - token which identifies a GETINFO parameter.
- item_entry - numeric value or text in ASCII (max. 255 character).

The response file will contain a GETINFO response entry for each defined GETINFO parameter.

2.8.2.4. GETID Response File Format

This response file type contains the results of the GETD request command. This command is used to obtain a is a list of all supported parameters of a specific machine. All entries are written in ASCII and consist of values delimited by list delimiter LD (see GETID request command) an terminated by the EOC (;) sequence.

The format of a GETID response entry is as follows:

{{param_id} LD {type of parameter} LD {integer digits} LD {fractional digits} LD {write permission} LD {unit} LD {description} ;} ...

where

- param_id (type APPLICATION_PARAMETER) - token of a defined parameter {e.g. ActTimCyc or @MyPrivateParameter, see Application Parameter Specification)
- type of parameter - data type of parameter, one character (A...alpha numeric, B...Boolean, N...numeric)
- integer digits (type NUM_CONST) - For numeric parameters, this will be set to the maximum number of digits in the integer portion of the numeric value. For character string paramters, this entry will indicate

the maximum number of characters used to represent the parameter. For Boolean values, this will always be set to 1.

- fractional digits (type NUM_CONST) **-** For numeric parameters, this will be set to the number of fractional digits used to represent the parameter. For character and Boolean parameters, this entry is always 0.
- write permission (type NUM_CONST) **-** If set to 1, this parameter may be modified by JOB command requests. If set to 0, this parameter may not be modified using JOB commands.
- unit **-** ASCII text in quotes, max. 255 characters, describing the units used to represent the value.
- description **-** ASCII text in quotes, max. 255 characters, describing the parameter itself.

2.8.2.5. General Information File Format

A General Information File type has been defined to provide a central access point for application programs to obtain a list of all network machines along with the location of each machine's SESSION file path entry. The file is to be named MACHINE.INI and consists of the following information:

List of available machines:

[MACHINES] EOL
{{n}={internal_machine_identification} EOL} ...

SESSION file path entry:

{[internal_machine_identification] EOL
SESSIONPATH={path} EOL}...

MAXSESSIONS={MaxSessions}EOL

where

- [MACHINES] **-** section for machine list.
- n **-** logical machine number (1..n for simple access).
- internal_machine_identification **-** logical name or number for machine identification in this file.
- {path} **-** the network path information representing the location of the machine's SESSION file. It can be a relative or absolute path. The path string is **not** enclosed with quotation marks.
- MAXSESSIONS= **-** set to the maximum number of communication sessions (used in creation of session request file names).

This file can also be used also for additional manufacture dependent information. For example, an entry might be made to indicate the type of underlying network such as TCP/IP or SERIAL.

The following is an example of a MACHINE.INI file:

```
[MACHINES]
1=MACHINE_1
2=MACHINE_2
3=MACHINE_3
4=MACHINE_4
```

```
[MACHINE_1]
SESSIONPATH=\\SV1\INTERFACE\MACH1
MAXSESSIONS=3
IPADDRESS=128.123.200.103             // manufacture dependent entry

[MACHINE_2]
SESSIONPATH=\\SV2\INTERFACE\MACH2
MAXSESSIONS=2
IPADDRESS =128.123.200.102            // manufacture dependent entry

[MACHINE_3]
SESSIONPATH=\\SV2\INTERFACE\MACH3
MAXSESSIONS=2
IPADDRESS =128.123.200.101            // manufacture dependent entry

[MACHINE_4]
SESSIONPATH=\\SV1\INTERFACE\MACH4
MAXSESSIONS=8
IPADDRESS =128.123.200.100            // manufacture dependent entry
```

2.8.3. Event Log Types and Token

The following event log types have been defined:

- Active machine alarms. The token for the EVENT command for reporting current active alarms is CURRENT_ALARMS
- Machine alarms event protocol. The token for the EVENT command is ALARMS
- Overall change event protocol, which contains all changes regarding to setup parameter, machine state, system state, etc. The token for the EVENT command is CHANGES.
  Note: Only set values are reported. A set value is a parameter with write permission.

3. Appendices

3.1. Abbreviations and Acronyms

SPI	Society of the Plastics Industry Inc.
EUROMAP	European Committee of Machinery Manufacturers for the Plastics and Rubber Industries
ASCII	American Standard Code for Information Interchange
OSI	Open System Interconnection
FCB	File Based Communication
CCP	Committee on Communication Protocol
IMM	Injection Mould Machine
LD	List delimiter
EOL	End of line
EOC	End of command

3.2. White Spaces and Special Characters

Character	**Hex Value**	**Description**
space	20	Blank space character.
carriage return <cr>	0D	Carriage return character.
line feed <lf>	0A	Line feed character.
tab	09	TAB character.

Table , Session and Presentation Layer White-Space characters

Comments can be inserted at any point in which white-space is valid.

Purpose	**Character**	**Description**
Comment	//	Comments are denoted by two consecutive forward slashes. All characters following this sequence to the end of line (EOL) is considered comment information and is to be ignored by the command processor (as if white-space)
EOL	<cr><lf> or <cr>	End of line character sequence
EOC	;	End of command
LD	,	List delimiter

Table , Session and Presentation Layer Special Characters

3.3. Communication Session Flow

3.3.1. Session Request Flow Chart

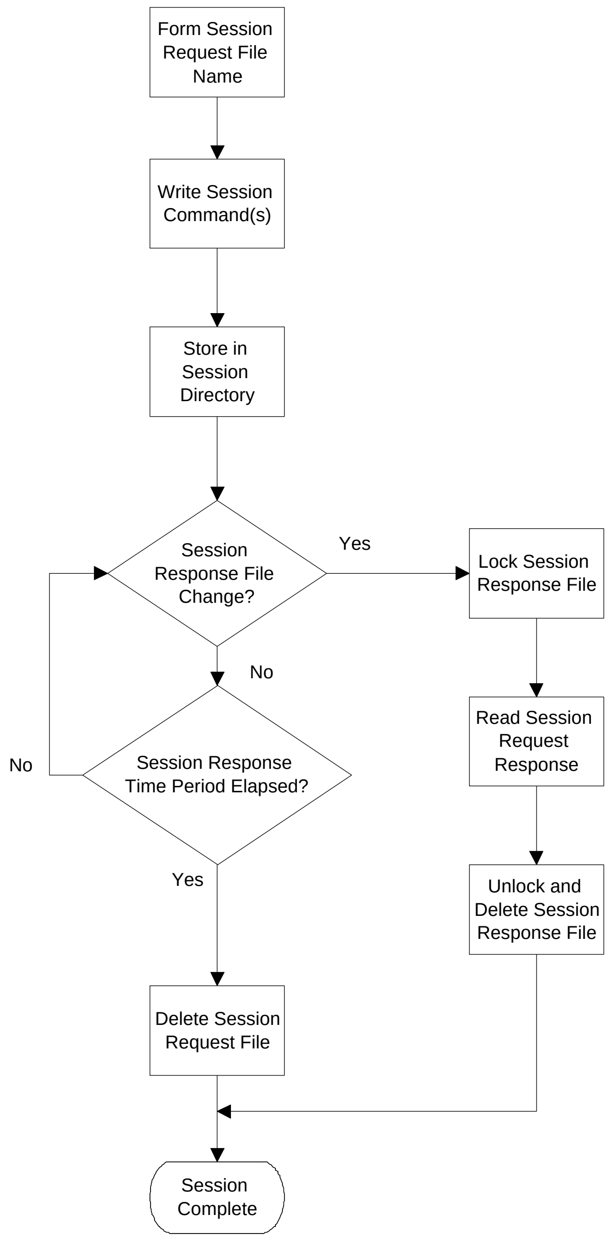
Form Session Request File Name
Write Session Command(s)
Store in Session Directory
Session Response File Change?
Yes
Lock Session Response File
No
Read Session Request Response
No
Session Response Time Period Elapsed?
Yes
Unlock and Delete Session Response File
Delete Session Request File
Session Complete

Figure , Session Request Flow Chart

3.3.2. Session Replay Flow Chart

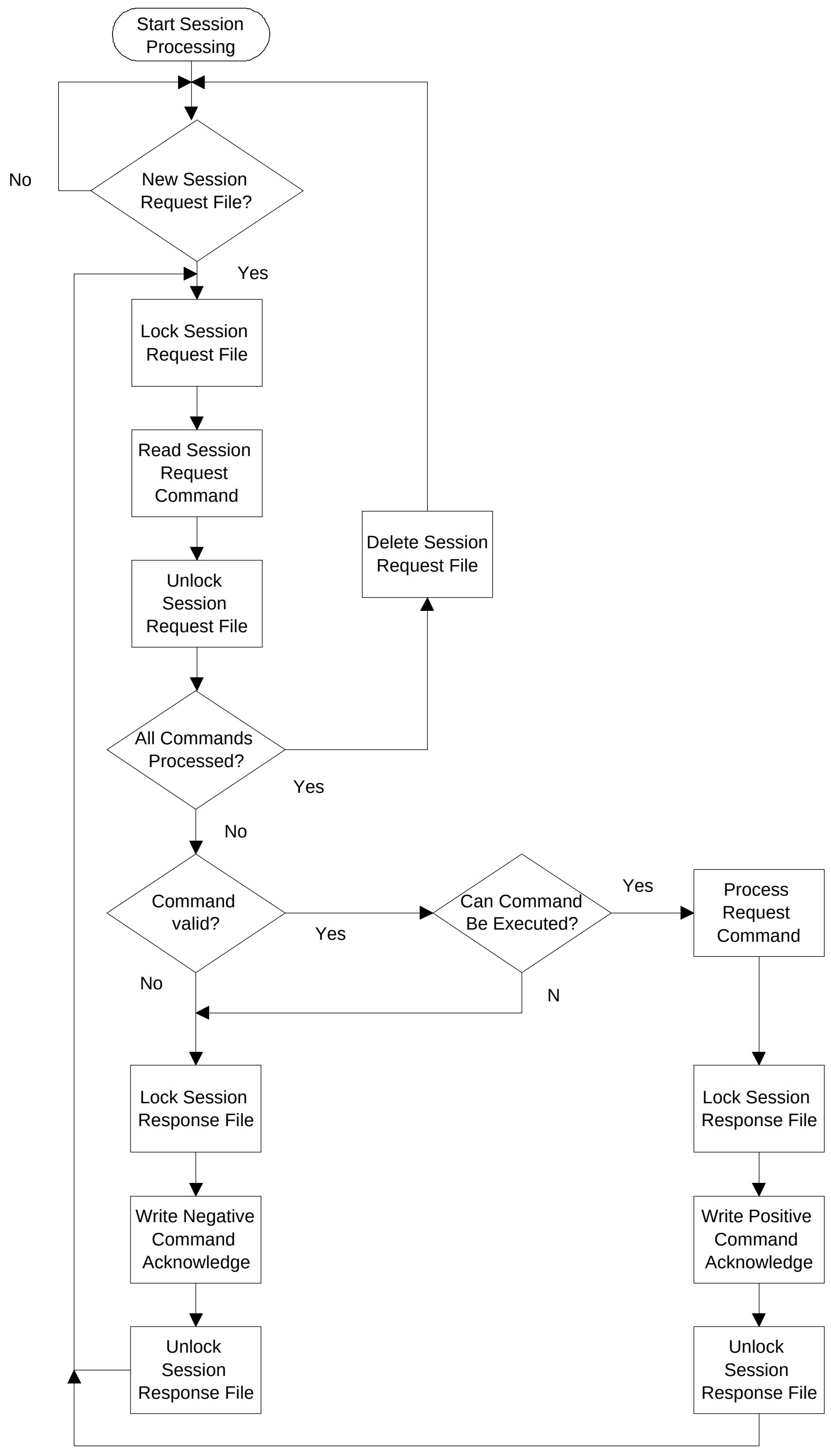
Start Session Processing
New Session Request File?
No
Yes
Lock Session Request File
Read Session Request Command
Unlock Session Request File
Delete Session Request File
All Commands Processed?
Yes
No
Command valid?
Yes
Can Command Be Executed?
Yes
Process Request Command
No
N
Lock Session Response File
Write Negative Command Acknowledge
Unlock Session Response File
Lock Session Response File
Write Positive Command Acknowledge
Unlock Session Response File

Figure , Session Replay Flow Chart

3.4. Communication Presentation Flow

Immediately after reading the request file, <PROCESSED "JOB command"> is responded, if the syntax of the commands is correct.

The PROCESSED keyword for each other following commands are written **after** the command is really processed.. If the command is terminated due to an error, the ERROR keyword is written as soon as the error occurs. If the command (eg. REPORT with STOP NEVER) never finishes, the PROCESSED keyword is never written.

This definition allows host computer really to recognize when the command is finished and it is not necessary to assume a timeout timer.

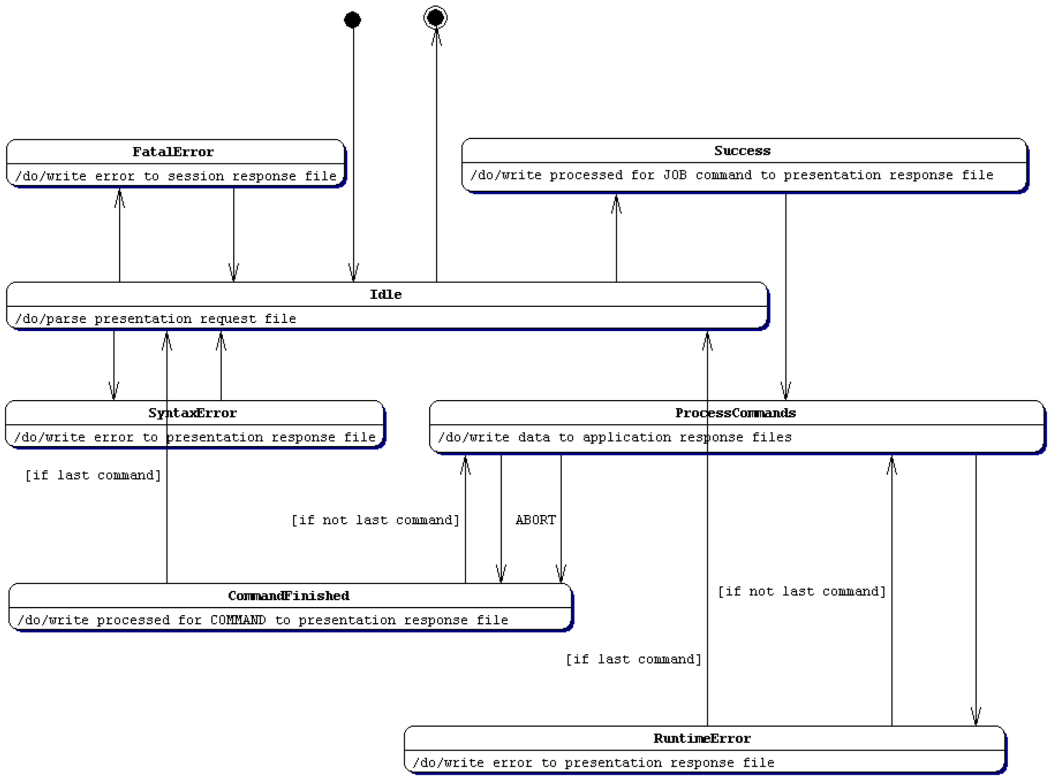
FatalError
/do/write error to session response file
Success
/do/write processed for JOB command to presentation response file
Idle
/do/parse presentation request file
SyntaxError
/do/write error to presentation response file
ProcessCommands
/do/write data to application response files
[if last command]
[if not last command]
ABORT
CommandFinished
/do/write processed for COMMAND to presentation response file
[if not last command]
[if last command]
RuntimeError
/do/write error to presentation response file

Figure , Session Request Flow Chart

3.5. Error Class Codes

Error information is provided to help identify various problems which may occur. Error information is classified according to the specific layer in which it was detected. The following chart identifies the various error classification codes.

Class	**Description**
01	Physical layer error codes.
02	Data link layer error codes.
03	Network layer error codes.
04	Transport layer error codes.
05	Session layer error codes.
06	Presentation layer error codes.
07	Generic application layer error codes.
08 – 99	Application specific error codes (registered by application type).

Table , Error Class Codes

3.6. Session Layer Error Codes

The following error codes are used by the CCP FBC Session Layer protocol. The codes are classified as ‘05’ Session layer error codes.

Code	**Description**
00000000	No error.
00000001	No response from network station.
00000002	Invalid syntax in session request command
00000003	Unable to create/open JOB response file.
00000004	Interface was started Note: Returned only from the first CONNECT command after the startup of the network station (machine). It means, that all running jobs, before shutdown, are lost.
00000005	Interface is busy.
00000006	Machine is offline. Note: In case of EXECUTE the presentation request file will be rejected.
00000007	Invalid syntax in presentation request JOB command Note: Cant recognize the presentation response file specification
00000008 - 00009999	Reserved
00010000 - 99999999	Open for manufacture specific error codes

Table , Session Layer Error Codes

3.7. Session Request Commands and Responses

The following sections list the Session Request File Commands and Responses.

3.7.1. Session Connection Verification Command

Command:	CONNECT
Format:	CONNECT;
Parameters:	none
Example:	`REQ_0002 CONNECT;   // verify connection`
Response Data:	Session layer error or completion message
Example:	`REQ_0002 PROCESSED;`

Description:

This command is used to verify the connection between network stations at the session layer. The requester station writes this command to the command file and then monitors the response file. The responding station, upon detecting this new request, processes the request at the session layer by writing a positive acknowledgment to the response file.

If the requesting station receives no response within the specified time-out period the requesting station should delete the session request file, to prevent the inadvertent processing of not processed session request files. If the requesting station receives a negative acknowledgment to the request, it must consider the station connection as invalid and reattempt the request at a later time. If the requesting station receives a positive response to the request it can consider the station as connected.

3.7.2. Session Execute Command

Command:	EXECUTE
Format:	EXECUTE {fspec};
Parameters:	
Fspec	A presentation layer or application layer command file specification.
Example:	`REQ_0003 EXECUTE “\\SV1\VOL1\MACH01\ANY.JOB”;` `// execute command file`
Response Data:	Session layer error or completion message
Example:	`REQ_0003 ERROR 00000001 Any 255 char message…;` - or - `REQ_0003 PROCESSED;`

Description:

This command is provided as a service to the presentation or application layers to request a network station to execute a command file. The command file itself is an upper layer entity and is meaningless at the session layer. The responsibility at the session layer is to inform the appropriate network station of the application layer entity to process.

The execution request is initiated by writing the execute command to the session request file. The command must provide an appropriate file specification of the network resource which contains the application layer file. This specification must include all information required to identify the storage device and file such as server name, drive name, directory name, and file name. The entire file specification must be representable as a variable length character string which is less than or equal to 255 characters in length.

The responding network station, upon detecting the execution request, will present the file specification information to the presentation layer for processing. If the upper layers accept the command file, it returns a positive acknowledgment to the session layer. If the upper layers refuse the command file, a negative acknowledgment is returned to the session layer.

{fspec} see chapter **File Specification Parameter.**

3.8. Presentation Layer Error Codes

The following error codes are used by the CCP FBC Presentation Layer protocol. The codes are classified as ‘06’ Presentation layer error codes.

Code	**Description**
00000000	No error.
00000001	Reserved.
00000002	Reserved.
00000003	Too many active JOBS.
00000004	Unable to create/open destination file.
00000005	Invalid REPORT command syntax.
00000006	Unknown REPORT parameter.
00000007	Too many active REPORTS.
00000008	Invalid EVENT command syntax.
00000009	Too many active EVENTS.
00000010	Unknown EVENT token.
00000011	Invalid ABORT command syntax.
00000012	Invalid UPLOAD command syntax.
00000013	Unable to create/open UPLOAD destination directory/file.
00000014	Invalid DOWNLOAD command syntax.
00000015	Unable to open DOWNLOAD source directory/file.
00000016	DOWNLOAD operation denied.
00000017	Invalid GETINFO command syntax.
00000018	Invalid GETID command syntax.
00000019	Invalid SET command syntax.
00000020	SET operation of parameter denied.
00000021	SET value out of range.
00000022	Unknown SET parameter.
00000023	Unsupported command.
00000024	Invalid START parameter syntax.
00000025	Invalid STOP parameter syntax.
00000026	Application parameter or constant parameter to long.
00000027	Invalid numeric format.
00000028	Invalid comparison operator.
00000029	Invalid time format.
00000030	Invalid date format.
00000031	End of command expected.
00000032	Unknown error.
00000033	REPORT with the same name and type is already running.
00000034	EVENT with the same name and type is already running.
00000035	DOWNLOAD is already running.
00000036	The specified command to abort, is not active.
00000037	Error during command processing.
00000038 - 00009999	reserved
00010000 - 99999999	Open for manufacture specific error codes

Table , Presentation Layer Error Codes

3.9. Presentation Request and Response Parameters

Many of the presentation layer request commands and command responses share various common parameter types and formats. This appendix provides a generic description of such common parameter formats. Each parameter type is given an appropriate name which is used by the presentation layer command and response descriptions to reference the parameter type. The following sections list the various common parameters.

3.9.1. File Specification Parameter

Parameter Type:	FSPEC
Format	{fspec}
Data Types	
{fspec}	VSTRING(255) **-** the file specification string
Example:	“\\sv1\vol1\mach03\get_temp.job” // absolute path spec.
	“g:\euro\mach01\get_temp.job // absolute path spec.
	“mach01\get_temp.job // relative path spec.

Description:

This parameter type is used to specify the location of a network file. The specification string must include all appropriate information required to locate (or create) the file on the network device. All strings are enclosed with quotation marks and are delimited to 255 chars.
In case of a relative path specification the whole path is the result of concatination of the absolute path of the file where the relative Path is specified and this relative path. The relative path specification is allowed in all file types.

Example: Session request file is located at:
“\\sv1\vol1\mach03\sess0001.req”.
The file contains:
00000001 EXECUTE “commands\report.job”
The location of this job is:
“\\sv1\vol1\mach03\commands\report.job ”

3.9.2. Time Specification Parameter

Parameter Type:	TIME_SPEC
Format	{hh:mm:ss}
Data Types	
{hh:mm:ss}	FSTRING(8) **-** the time specification string in 24-hour local time where hh is hours, mm is minutes, and ss is seconds.
Example:	`13:00:00   // specify 1:00PM`

Description:

This parameter type is used to specify a time value. Time values are represented as an 8 character fixed length string. Time values are always represented in 24 hour format. The specification ***does not*** specify a time zone standard (time values are entered according to the local standard time).

Valid time specification values are:

- hours from 00 to 23
- minutes from 00 to 59
- seconds from 00 to 59

3.9.3. Date Specification Parameter

Parameter Type:	DATE_SPEC
Format	{yyyymmdd}
Data Types	
{yyyymmdd}	FSTRING(8) - the date specification string in the specified format/order where mm indicates the month, dd indicates the date, and yy indicates the last two digits of the year.
Example:	`19970101   // New Year's 1997`

Description:

This parameter type is used to specify a date value. Date values are represented as an 8 character fixed length string. Date values are always represented in the specified format.

Valid date specification values are:

- months from 01 to 12
- date from 01 to 31 (dependent on month)
- year from 0000 - 9999

3.9.4. Comparison Operator Specification

Parameter Type:	COMPARE_OP
Format	{operator token}
Data Types	
=	The = token for equality comparison
!=	The != token for non-equality comparison
>	The > token for "a is greater than b" comparison
<	The < token for "a is less than b" comparison
>=	The >= token for "a is greater than or equal to b" comparison
<=	The <= token for "a is less than or equal to b" comparison
Example:	`ActTimCyc >= 1.0  // check if parm > const`

Description:

This parameter type is used to specify a comparison operator. The context of the comparison is dependent on the data types of the items to be compared.

3.9.5. Constant Parameter Specification

Parameter Type:	NUM_CONST | STRING_CONST | BOOL_CONST
Format	{type const}
Data Types	
{num_const}	A numeric constant.
{string_const}	A string constant
{bool_const}	A boolean constant

Example:

```
127.4          // numeric constant
“anystring“    // string const
0              // boolean const
```

Description:

This parameter type is used to specify various constants. The constants are specified as shown according to the specific data types (see section Physical Data Formats):

- Numeric values: Integer or fractional values, max. 16 digits exclusive of decimal point. Data type :NUMERIC(n).
- Alpha numeric: ASCII text in quotes with max. 255 characters. Data type: VSTRING(255).
- Boolean values : Integer value,. 1 digit, where the value 0 corresponds to FALSE and a value not 0 corresponds to TRUE. Data type :NUMERIC(1).

3.9.6. Application Parameter Specification

Parameter Type:	APPLICATION_PARAMETER
Format	{parm_id}
Data Types	
{param_id}	The token which identifies the application layer parameter, length max. 255 characters, case sensitive, no white-spaces.
Example:	`ActTimCyc // app parameter ID`

Description:

This parameter type is used to specify all application layer parameters. The parameters are specified as tokens which are meaningful at the application layer. The parameter ID specified must be valid according to the required data type of the context in which the parameter is used. Each specific command which requires a parameter ID will specify the appropriate parameter data type.

Supported data types (see section Physical Data Formats).

- Numeric values (Data type: NUMERIC(n)): Integer or fractional values, max. 16 digits exclusive of decimal point.
- Alpha numeric (Data type: VSTRING(255)): ASCII text in quotes with max. 255 characters.
- Boolean values (Data type: NUMERIC(1)): Integer value,. 1 digit, where the value 0 corresponds to FALSE and a value 1 corresponds to TRUE.

3.9.7. Start Specification Parameter

Parameter Type:	START_SPEC
Format	START [IMMEDIATE | TIME>= {time_spec} [{date_spec}] | {parm_id} {compare_op} {num_const} | {string_const}]
Data Types	
START	The keyword START
IMMEDIATE	The keyword IMMEDIATE indicating to start the function immediately
TIME>=	The keyword TIME>= followed by a time specification and optional date specification to indicate to start the function at or after the specified time/date. If no date specification is supplied the current date is used.
{parm_id}	A valid application layer parameter ID of a numeric parameter used to compare against a constant. When the comparison becomes TRUE the function is started.
{compare_op}	A valid comparison operator token.
{num_const}	A valid numeric constant.
{string_const}	A valid string constant.

Example:	`START IMMEDIATE // start now!`
	`START TIME>= 08:00:00 19970101 // start then!`
	`START ActTimCyc > 2.00 // start if!`

Description:

This parameter type is used to specify when to start a function. The specification allows three basic formats:

1. Start immediately upon execution of the command
2. Start at the specified time and date
3. Start when the specified parameter meets the specified condition

When an application layer parameter ID is specified, it must be of an appropriate data type to perform a numeric comparison. The responding session layer will query its application layer for the actual parameter value.

3.9.8. Stop Specification Parameter

Parameter Type:	STOP_SPEC
Format	STOP [NEVER | TIME>= {time_spec} [{date_spec}] | {parm_id} {compare_op} {num_const} | {string_const}]
Data Types	
STOP	The keyword STOP
NEVER	The keyword NEVER indicating that the stop mechanism is not specified by this clause (the operation will continue forever until some other operation cancels it)
TIME>=	The keyword TIME>= followed by a time specification and optional date specification to indicate to stop the function at or after the specified time/date. If no date specification is supplied the current date is used.
{parm_id}	A valid application layer parameter ID of a numeric parameter used to compare against a constant. When the comparison becomes TRUE the function is stoped.
{compare_op}	A valid comparison operator token.
{num_const}	A valid numeric constant.
{string_const}	A valid string constant.

Example:	`STOP NEVER // don't stop!`
	`STOP TIME>= 08:00:00 19970101 // stop when!`
	`STOP ActTimCyc > 2.00 // stop if!`

Description:

This parameter type is used to specify when to stop a function. The specification allows three basic formats:

1. Do not stop based on this clause. This format is used when the function is to run until stopped by some other mechanism.
2. Stop at the specified time and date
3. Stop when the specified parameter meets the specified condition

When an application layer parameter ID is specified, it must be of an appropriate data type to perform a numeric comparison. The responding session layer will query its application layer for the actual parameter value.

3.10. Presentation Request Commands and Responses

3.10.1. Presentation Job Request Command

Command: JOB

Format: JOB {name} RESPONSE {fspec};

Parameters:

JOB	The keyword JOB indicating the start of the JOB command.
name	A VSTRING(16) parameter (without quotation marks) which contains any character information used to identify the job request. White-space characters not allowed.
RESPONSE	The keyword RESPONSE used to indicate that a presentation request response file specification string will follow.
fspec	A presentation layer file specification which identifies the JOB response file. The file is created by the responder and all request command responses are written to this file.

Example:
```
JOB read_mold RESPONSE
”\\sv1\vol1\mach01\any.log”;
                           // job specification
```

Response Data: Presentation layer error or completion message

Example:
```
COMMAND 1 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```
- or -
```
COMMAND 1 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to specify the start of a presentation job. The command ***must*** be the first command of the presentation request command file. The command is used to specify the file specification of the file used to write all job command responses. If the responding station cannot open the response file, or if a syntax error is detected in the job command, processing of the job file is aborted and a negative acknowledgment is returned to the session layer.

Each Job Request File can contain ***only a single*** command, excluding the SET command. Multiple SET commands ***are valid*** in a single Job Request File to allow multiple setpoint values to be set using a single Job Request.

3.10.2. Report Generation Request Command

Command: REPORT

Format:
```
REPORT {name} [APPEND | REWRITE] {fspec} {start_spec}
{stop_spec}
[IF [NOT] {parm_id} [{compare_op} {num_const} |
{string_const} | {bool_const} | CHANGE]]
[CYCLIC [TIME {time_const} | SHOT {num_const}]
[SAMPLES {num_const}]
[SESSIONS {num_const}]]
PARAMETERS
   {parm_id} | TIME | DATE | COUNT
     [,{{parm_id} | TIME | DATE | COUNT }…] ;
```

Parameters:

REPORT	The keyword REPORT indicating the start of the REPORT command.
name	The name of the report. This VSTRING(16) parameter (without quotation marks) is used to identify the report. The report name

	must be unique from that of any other active report at the responder station.
APPEND	The optional keyword APPEND. If specified, the report data is appended to the end of any existing report file. If omitted, any existing report file is deleted. Note that this option is not valid with the REWRITE option. If the response was deleted by the host, the header line has to written again.
REWRITE	The optional keyword REWRITE. If specified, any existing report data is overwritten with new data each cycle or time period. If omitted, all new data is added to the end of the report. Note that this option is not valid with the APPEND option.
fspec	A network file specification which specifies the report file. All reported information is written to this file.
start_spec	A valid START specification clause used to indicate when report data generation will be started.
stop_spec	A valid STOP specification clause used to indicate when report data generation will be stopped.
IF [NOT]	The optional keyword IF indicating the start of a conditional reporting clause. This clause is used to enable or inhibit report record generation. The IF clause will inhibit record generation if the conditional clause is FALSE while the IF NOT clause will inhibit record generation if the conditional clause is TRUE. If no conditional reporting clause is specified report record generation is always enabled.
{parm_id} {compare_op} {num_const} {string_const} {bool_const}	This condition is specified to enable or inhibit report record generation based on the comparison of an application parameter value to a numeric constant, to a string constant or a boolean constant. If no compare operator is defined, the value of the {param_id} (type APPLICATION_PARAMETER) is interpreted as a boolean expression with the following meanings: {bool_const}: 0 is false and ≠ 0 is true {num_const}: 0 is false and ≠ 0 is true {string_const}: empty string (“”) is false and otherwise true
CHANGE	This condition is specified to enable or inhibit report record generation based on whether the specified application parameter has changed since the last record was recorded. It is up to the application layer to report whether the parameter has changed when queried by the presentation layer.
CYCLIC	This optional keyword is used to specify whether the report parameters are recorded only once or whether they are recorded on a cyclic basis. If the CYCLIC keyword is omitted, the data is recorded only once. If the CYCLIC keyword is present, the data is recorded by default every machine cycle. If an optional TIME or SHOT clause is provided the data is recorded according to the clause specification. Note that it is up to the application layer to specify the definition of a “machine cycle”. Note: One record corresponds to one line in the answer file.
TIME {time const}	The optional CYCLIC clause TIME keyword indicating that data is to be recorded based on a time interval. The time constant value is specified as a FSTRING(8) in “hh:mm:ss” format.
SHOT {num_const}	The optional CYCLIC clause SHOT keyword indicating that data is to be recorded based on a shot count interval. The numeric constant specifies the number of shots in each recording interval.
SAMPLES	The optional SAMPLES keyword specifies the number of

{num_const}	successive recording samples to issue. If omitted, a single recording sample is issued each cyclic period. If present, the specified number of samples are recorded each cyclic period.
SESSIONS {num_const}	The optional SESSIONS keyword specifies the total number of recording sessions to report. If omitted, recording continues until the STOP clause activates or the report is aborted. If specified, recording is stopped once the specified number of recording sessions have been completed.
PARAMETERS	The keyword PARAMETERS indicating the start of the list of parameters to be reported. At least one parameter is required. A parameter may be an application layer parameter ID token or a pseudo parameter token. Each parameter is separated with a comma (,) character. The order of the parameters specifies the order of the data in the report records.
{parm_id}	The application layer parameter ID token which specifies a parameter to be monitored.
TIME	A pseudo parameter used to specify that the current time be output as a data value to the report. The time value is output as a FSTRING(8) string in standard TIME format (HH:MM:SS).
DATE	A pseudo parameter used to specify that the current date be output as a data value to the report. The date value is output as a FSTRING(8) string in standard DATE format (YYYYMMDD).
COUNT	A pseudo parameter used to specify that the current report record number be output as a data value to the report. Each time a REPORT command is processed its record counter is reset to 1. After each data record is recorded to the report file, this counter is incremented. The COUNT is stored as a ULONG data value.

Example:

```
REPORT REP_CYC
“\\sv4\vol1\mach0\cycdata.rep”
   START IMMEDIATE
   STOP NEVER
   IF NOT ActCntCycRej CHANGE
   CYCLIC TIME 00:00:15
   SAMPLES 5
   SESSIONS 150
   PARAMETERS
    COUNT, TIME, DATE, ActTimCyc, ActCntCyc;
```

Response Data: Presentation layer error or completion message

Example:

```
COMMAND 2 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```

- or -

```
COMMAND 2 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to generate an application data report. The command provides various required and optional parameters which are used to specify how the report is generated. Each report is given a unique 8 character name used to identify the report. Following the report name is a file specification string which identifies the file used to contain the report data. If the APPEND keyword is specified, new report data is appended to the end of the file. If the APPEND keyword is not specified, any existing file data is deleted.

Start and stop specification parameters are used to start and stop report data generation. If the STOP NEVER specification is used, report data generation will continue until an ABORT command is issued or an error is detected.

A conditional reporting clause can be specified to indicate when parameter values are to be recorded. If specified, a conditional clause is used to compare a parameter to a constant or to detect if a parameter has changed. It is up to the application layer to provide these functions. If no conditional reporting clause is provided parameter recording is always enabled.

Data can be recorded in once-only mode or cyclic mode. The default once-only mode will record the data in a single session and then terminate (regardless of any STOP clause). The CYCLIC keyword is used to specify cyclic mode recording. The recording can be based on time or a shot/cycle count. When based on time, recording is started each time the specified time interval expires. When based on shot/cycle count, recording is started each time the specified number of shots are completed.

The optional SAMPLES keyword is used to specify the amount of parameter samples to be recorded each recording session. By default, a single sample of each parameter is recorded. If the SAMPLES keyword is used, the specified number of samples are recorded each session. Each sample is recording at the completion of a machine shot/cycle. Note that it is up to the application layer to define what a shot/cycle is.

The optional SESSIONS keyword is used to specify the total number of recording sessions to process. If omitted, recording will continue until any STOP clause activates or the report is aborted. If specified, recording is stopped if the STOP clause activates, the report is aborted, or the specified number of sessions are recorded (whichever occurs first).

The PARAMETERS keyword is used to specify the start of the recording parameter list. This list consists of application layer parameter ID tokens and the special pseudo tokens TIME, DATE, and COUNT. Each parameter is separated by a comma. The last parameter in the list is ended with the command end character (;) (which also terminates the REPORT command). Each recording session, the application supplies the parameter data values for recording.

Parameters are recording in the same order as specified in the PARAMETERS list. Parameter values are recorded in the appropriate data formats as defined by the session layer and set by the application layer. Each value is separated with a comma (,) character. The last value in the recording list is terminated with an EOL sequence.

CYCLIC SHOT 10
SAMPLES 2
SESSIONS 20

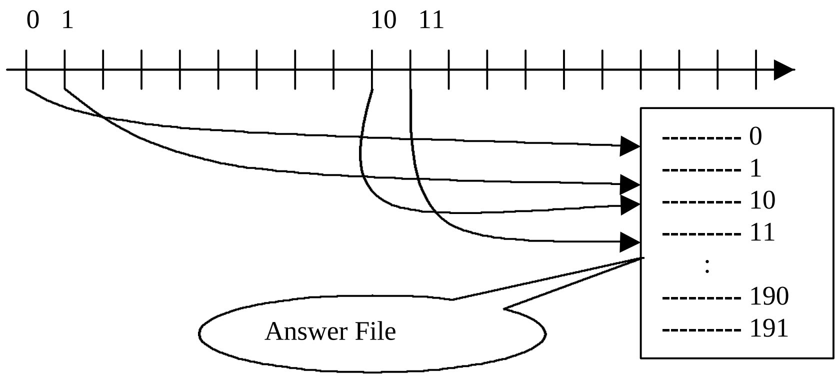

Figure , CYCLIC Example Flow Chart

3.10.3. Event Log Request Command

Command:	EVENT
Format:	EVENT {name} {type} [APPEND | REWRITE] {fspec} {start_spec} {stop_spec};
Parameters:	
EVENT	The keyword EVENT indicating the start of the Event Log request.
name	The name of the event. This VSTRING(16) parameter (without quotation marks) is used to identify the event. The event name must be unique from that of any other active event at the responder station.
type	A token which specifies the type of event to log. The actual event types are specified at the application layer.
APPEND	The optional keyword APPEND. If specified, the event data is appended to the end of any existing event file. If omitted, any existing event file is deleted.
REWRITE	The optional keyword REWRITE. If specified, any existing report data is overwritten with new data each cycle or time period. If omitted, all new data is added to the end of the report. Note that this option is not valid with the APPEND option.
fspec	A network file specification which specifies the event file. All reported information is written to this file.
start_spec	A valid START specification clause used to indicate when event data generation will be started.
stop_spec	A valid STOP specification clause used to indicate when event data generation will be stopped.

Example:

```
EVENT myEventAlarms ALARMS APPEND
“\\sv1\vol1\mach0\log\mach0.alr”
   START IMMEDIATE
   STOP TIME>= 08:00:00;
```

Response Data: Presentation layer error or completion message

Example:

```
COMMAND 71 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```

- or -

```
COMMAND 71 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to generate an event log. The command differs from the REPORT command in that it is designed to log a single type of data which occurs as an asynchronous event. The command provides parameters to specify what type of event is to be logged, when to start event logging, and when to stop event logging.

The type of event to log is specified using an application layer event ID token. The token is passed to the application layer to register the event for logging. When an event occurs, the application layer must pass this parameter ID, along with the event record data, to the presentation layer for recording to the event log file. The actual types of events supported, along with the type and format of the event record data, is defined by the specific application layers.

Following the event type is a file specification string which identifies the file used to contain the event log data. If the APPEND keyword is specified, new event data is appended to the end of the file. If the APPEND keyword is not specified, any existing file data is deleted at the start of event logging.

Start and stop specification parameters are used to start and stop event data generation. If the STOP NEVER specification is used, report data generation will continue until an ABORT command is issued, command request file is deleted or an error is detected.

3.10.4. Abort Request Command

Command: ABORT

Format: ABORT
ALL [JOBS | REPORTS | EVENTS] |
JOB {name} |
REPORT {name} |
EVENT {name} ;

Parameters:

ABORT	The keyword ABORT indicating the start of the Abort request.
ALL [JOBS | REPORTS | EVENTS]	The keyword ALL is used to abort one or more functions at the same time. When used alone, all active jobs (excluding the current job executing this command), reports, and events are aborted. When used with the keywords JOBS, REPORTS, or EVENTS, all active operations of the specified type are aborted.
JOB {name}	The keyword JOB indicating that a JOB is to be aborted. The {name} data must be the name specified in the JOB command of the job to abort.
REPORT {name}	The keyword REPORT indicating that a REPORT is to be aborted. The {name} data must be the name specified in the REPORT command of the report to abort.
EVENT {name}	The keyword EVENT indicating that an EVENT is to be aborted. The {name} data must be the event name specified in the EVENT command of the event log to abort.

Example:
```
ABORT REPORT REP_CYC;    // abort the report
```

Response Data: Presentation layer error or completion message

Example:
```
COMMAND 9 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```
- or -
```
COMMAND 9 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to abort active presentation and application layer processes. The abort command can be specified in various formats to allow one or more active functions to be aborted. When an ABORT is processed, all appropriate files and functions must be closed and terminated according to the requirements of the specific operations being aborted.

3.10.5. Upload Request Command

Upload is defined as direction from machine to host.

Command: UPLOAD

Format: UPLOAD {fspec} ACTIVE | {av_fspec} | {gen_keyword} {start_spec};

Parameters:

UPLOAD	The keyword UPLOAD indicating the upload of a setup dataset.
fspec	A network file specification which specifies the dataset file or directory. All reported information is written to this file or directory.
ACTIVE	The keyword ACTIVE indicating the upload of the active setup

	dataset from the machine.
av_fspec	A file specification which specifies a dataset in the machine archives indicating the upload of this dataset.
gen_keyword	A manufactory specific keyword which specifies a proprietary file transfer.
start_spec	A valid START specification clause used to indicate when data set upload will be started.

Example:

```
UPLOAD “\\sv1\vol1\data\mach_00.stp”
  ACTIVE
  START IMM_MOULDNUMBER = 12345678; //
```

Response Data: Presentation layer error or completion message

Example:

```
COMMAND 7 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```

- or -

```
COMMAND 7 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to upload a setup data set from a specific machine and store it to the file {fspec}.

3.10.6. Download request Command

Download is defined as direction host to machine.

Command: DOWNLOAD

Format: DOWNLOAD {fspec} ACTIVE | {av_fspec} | {gen_keyword} {start_spec};

Parameters:

DOWNLOAD	The keyword DOWNLOAD indicating the download of a setup dataset.
fspec	A network file specification which specifies the dataset file or directory. All reported information is written to this file or directory.
ACTIVE	The keyword ACTIVE indicating the download setup dataset as active dataset in the machine.
av_fspec	A file specification which specifies a dataset in the machine archives.
gen_keyword	A manufactory specific keyword which specifies a proprietary file transfer.
start_spec	A valid START specification clause used to indicate when data set download will be started.

Example:

```
DOWNLOAD “\\sv1\vol1\data\mach_01.stp”
  ACTIVE
  START IMMEDIATE; //
```

Response Data: Presentation layer error or completion message

Example:

```
COMMAND 8 ERROR 06 00000001 “Any 255 char
message…“ 19971212 13:30:59;
```

- or -

```
COMMAND 8 PROCESSED “Any 255 char response
data ...“ 19971212 13:40:31;
```

Description:

This command is used to download a setup data set to a specific machine and store it to the file {fspec}. If the ARCHIVES keyword is not specified the active data set otherwise the specified data set from the archives is transferred.

3.10.7. GETINFO request Command

Command:		GETINFO
Format:		GETINFO {fspec};
Parameters:		
	GETINFO	The keyword GETINFO indicating an upload of a hard- and software information of a machine.
	fspec	A network file specification which specifies the response file. All reported information is written to this file.
Example:		`GETINFO “\\sv\vol1\data\mach_00.inf”; //`
Response Data:		Presentation layer error or completion message
Example:		`COMMAND 9 ERROR 06 00000001 “Any 255 char message…“ 19971212 13:30:59;` - or - `COMMAND 9 PROCESSED “Any 255 char response data ...“ 19971212 13:40:31;`

Description:

This command is used to obtain information about the hardware and software from an IMM and write it to the file {fspec}.

The following information is written to the specified application layer file in response to this command:

Item Id (case sensitiv)	**Description**
MachVendor	Name of machine vendor, ASCII text in quotes, max. 255 character
MachNbr	Machine serial number, ASCII text in quotes, max. 255 characters
MachDesc	Machine description - ASCII text in quotes, max. 255 characters
ContrType	Controller type – ASCII text in quotes, max. 255 characters.
ContrVersion	Controller version – ASCII text in quotes, max 255 characters.
Version	Software version – version of EUROMAP - SPI definition.
MaxJobs	Max. count jobs – numeric value in ASCII.
MaxEvents	Max. active events for each type – list of type token with numeric value in ASCII.
DownloadTypes	List of manufatcory specific keywords which specify a properitary download.
UploadTypes	List of manufatcory specific keywords which specify a properitary upload.
MaxReports	Max. active reports – numeric value in ASCII.
MaxArchives	Max. archives – numeric value in ASCII (0 .. achives not supported)
InjUnitNbr	Number of injection units – numeric value in ASCII
MaterialNbr	Number of materials – numeric value in ASCII
CharDef	Character definition, DOS-Codepage – ASCII text in quotes, max 255 characters.
MaxSessions	Defines the maximum number of sessions supported by a machine. Also used in the formation of valid Session Request File names.
ActiveJobs	A list of all active jobs.
ActiveReports	A list of all active reports.
ActiveEvents	A list of all active event logs.

Format specification:

MachVendor LD ”{string_const}“ ;
MachNbr LD ”{string_const}“ ;
MachDesc LD ”{string_const}“ ;
ContrType LD ”{string_const}“ ;
ContrVersion LD ”{string_const}“ ;
Version LD ”{string_const}“ ;
MaxJobs LD {num_const} ;
MaxEvents LD CHANGES {num_const} CURRENT_ALARMS {num_const} ALARMS {num_const};
DownloadTypes LD {gen_keyword} ...;
UploadTypes LD {gen_keyword} ...;
MaxReports LD {num_const} ;
MaxArchives LD {num_const} ;
InjUnitNbr LD {num_const} ;
MaterialNbr LD {num_const} ;
CharDef LD ”{string_const}“ ;
MaxSessions LD {num_const} ;
ActiveJobs LD {“{job_name}“ “{job_fspec}“ “{response_fspec}“} ... ;
ActiveReports LD {“{report_name}“ “{job_fspec}“ “{response_fspec}“} ... ;
ActiveEvents LD {“{event_name}“ “{event_type}“ “{job_fspec}“ “{response_fspec}“} ... ;

Example:

```
MachVendor,         "XYZ Machinery";
MachNbr,            "65535";
MachDesc,           "?";
ContrType,          "XYZ 3000";
ContrVersion,       "2.xx";
Version,            "1.03";
MaxJobs,            99;
MaxEvents,          CHANGES            2
                    CURRENT_ALARMS     1
                    ALARMS             0;
MaxReports,         99;
MaxArchives,        0;
InjUnitNbr,         ;
MaterialNbr,        ;
CharDef,            "850";
MaxSessions,        99;
ActiveJobs,
 "report"
  "\\emspc477\vol1\e63\machines\65535\p\report.job"
  "\\emspc477\vol1\e63\machines\65535\p\report.dat"
 "getinfo"
  "\\emspc477\vol1\e63\machines\65535\p\getinfo.job"
  "\\emspc477\vol1\e63\machines\65535\p\getinfo.dat";
ActiveReports,
 "status"
  "\\emspc477\vol1\e63\machines\65535\p\report.job"
  "\\emspc477\vol1\e63\machines\65535\p\report.dat";
ActiveEvents,
"mychanges" CHANGES
  "\\emspc477\vol1\e63\machines\65535\p\eventchanges.job"
  "\\emspc477\vol1\e63\machines\65535\p\eventchanges.dat"
"yourchanges" CHANGES
  "\\emspc477\vol1\e63\machines\65535\p\eventchanges2.job"
  "\\emspc477\vol1\e63\machines\65535\p\eventchanges2.dat";
```

3.10.8. GETID request Command

Command:	GETID
Format:	GETID {fspec};
Parameters:	
GETID	The keyword GETID indicating a upload of all available variables
fspec	A network file specification which specifies the response file. All reported information is written to this file.
Example:	`GETID “\\sv1\vol1\data\mach_00.id”;`
Response Data:	Presentation layer error or completion message

Example:	`COMMAND 18 ERROR 06 00000001 “Any 255 char message…“ 19971212 13:30:59;`
	- or -
	`COMMAND 18 PROCESSED “Any 255 char response data ...“ 19971212 13:40:31;`

Description:

This command is used to get all available variables from a IMM and store it to the file {fspec}. The file contains the following information:

Parameter name - max. 255 character, no white space
Parameter type - one character, where
  A .. alpha numeric value,
  N .. numeric value,
  B .. boolean value (0..false, != 0..true)
Integer digits - number of integer digits
Fractional digits - number of fractional digits
Write permission - one character (0 .. update not allowed, 1 .. update allowed)
Unit - ASCII text in quotes, max. 255 character
Description - ASCII text in quotes, max. 255 character

3.10.9. SET request Command

Command:	SET
Format:	SET {parm_id} Val;
Parameters:	
SET	The keyword SET indicating to set a variable in the IMM
{parm_id}	The application layer parameter ID token which specifies a parameter to be monitored.
Val	Value of the variable.
Example:	`SET SetTimMach 08300020000627;`
Response Data:	Presentation layer error or completion message

Example:	`COMMAND 21 ERROR 06 00000001 “Any 255 char message…“ 19971212 13:30:59;`
	- or -
	`COMMAND 21 PROCESSED “Any 255 char response`

```
            data ...“ 19971212 13:40:31;
```

Description:

This command is used to set a variable to a specific machine.

3.11. IMM Device Token Summary

The following section provides a list of the defined parameter tokens for the Injection Molding Machine application layer. The tokens describe both setup and actual parameters and have been created according to the token naming conventions previously described.

When creating vendor specific tokens outside of the tokens defined by the committee, the conventions described here should be followed. To identify the token as a vendor specific token, the at symbol '@' is used as a prefix to the token symbol.

Notes:

ASC string data is represented in the FORMAT columns as VSTRING(n) where VSTRING indicates a variable length ASCII string enclosed in quotation marks with a maximum length of n characters (excluding the quotation marks). The ASCII string can include the space character and contains ASCII code values based on the current code page in use by the machine.

Character field data is represented in the FORMAT columns as CHAR(n) where CHAR indicates a fixed length character string of n characters. The character fields are not enclosed in quotes. Each position within the field is given a set of valid characters for that position. The character values are always referenced according to U.S. code page 0.

Numeric data is represented in the FORMAT columns are NUMERIC. The numeric data is output as ASC text in normalized or scientific formats.

The **Set** and **Act** prefixes for command tokens are used to identify SETPOINT and ACTUAL machine values. These keywords DO NOT indicate whether a particular token can be READ or WRITTEN.

Array data is expressed with each dimension enclosed within square brackets, such as ***SetDescMat[2,1]***. The dimension brackets are always required and multiple dimensions are separated with commas within the single pair of brackets. All dimensions are numbered starting with 1 and an asterisk '*' can be used to specify all valid numbers for a particular dimension where applicable. The GETID command returns the maximum number of entries for all dimensional tokens.

Changes from previous version noted in italics.

Token	Req	Format	Units	Notes
SetDescMach	No	VSTRING(256)	n/a	Customer Device Description. Can be set to any text string to help identify the machine to the customer.
SetTimMach	Yes	CHAR(14)	n/a	Clock Synchronization – 14 character field formatted as follows: HHMMSSYYYYMMDD Where HH: The hours value from 00 to 23 MM: The minutes value from 00 to 59 SS: The seconds value from 00 to 59 YYYY: The year MM: The month from 01 to 12 DD: The date from 01 to 31
ActStsMach	Yes	CHAR (5)	n/a	Actual Machine Status. Each character field position is used as follows: Pos. 1: Machine Status 0: Machine is running (powered on) 1: Machine is not running (powered off) Pos. 2: Machine Mode A: Automatic mode selected S: Semi-automatic mode is selected M: Manual mode is selected U: Setup mode is selected H: Hold to Run mode is selected C: Commissioning / Maintenance mode is selected 0: Unknown state currently selected I: Idle state currently selected Pos. 3: Assist Call 0: No assistance is required 2: Assistance is required Pos. 4: Bad Part (cycle by cycle, is re-examined each cycle) 0: Last cycle not bad 1: Last cycle bad Pos. 5: Active Alarm 0: No active alarms 1: Alarms are active
ActStsCyc	No	VSTRING(256)	n/a	Actual Cycle Status. Text string describes reason machine is not currently in cycle.
SetCntCyc	No	NUMERIC	Cycles	Number of Machine Cycles Requested for Prod Run
ActCntCyc	Yes	NUMERIC	Cycles	Actual Cycle Count
ActCntCycRej	No	NUMERIC	Cycles	Production Rejects - Process Control
ActCntPrtRej	No	NUMERIC	Parts	Production Rejects Parts
SetDescJob	No	VSTRING(256)	n/a	Job Name / Description
SetDescOp	No	VSTRING(256)	n/a	Operator ID
SetDescPrt	No	VSTRING(256)	n/a	Part Name / Description
SetDescMld	No	VSTRING(256)	n/a	Mold or Tool Name / Description
SetDescMat[InjUnit, Material]	No	VSTRING(256)	n/a	Material Name - 1 or more entries for each injection unit. Dimensional limits set by machine.
SetDescMatLot[InjUnit, Material]	No	VSTRING(256)	n/a	Material Lot Number - 1 or more entries for each injection unit. Dimensional limits set by machine.
SetRecMld	No	VSTRING(256)	n/a	Mold Setup (Recipe) File Name
SetCntMld	No	NUMERIC	Count (Cavities)	Set Mold Count - Number of Cavities Run
ActCntMld	No	NUMERIC	Count (Cavities)	Act Mold Count - Number of Cavities Run
SetRecMldNxt	No	VSTRING(256)	n/a	Setup (Recipe) File Name of Next Mold to be Run
SetCntPrtBox	No	VSTRING(256)	n/a	Part Box Count
SetCntPrt	No	NUMERIC	Count	Piece Counter
ActCntPrt	No	NUMERIC	Count	Piece Counter

Table , Machine Status Tokens

Token	Req	Format	Units	Notes
ActCfgBrl[InjUnit]	No	CHAR(1)	n/a	Barrel Configuration - Active Barrels. Single character field for each injection unit. '0' indicates barrel is OFF, '1' indicates barrel is ON.
SetDescBrlZn[InjUnit, Zone].	No	VSTRING(256)	n/a	Barrel Zone Description / Name. Maximum number of injection units and zones specified by machine.
SetCfgBrlZn[InjUnit, Zone]	No	CHAR(1)	n/a	Barrel Temperature Zone Configuration. Single character field for each injection unit barrel zone as follows: O: Barrel Zone is OFF 0: Barrel Zone is Not Supported A: Barrel Zone is in AUTO mode T: Barrel Zone is in TUNING mode S: Barrel Zone is in STANDBY mode M: Barrel Zone is in MANUAL mode
SetTmpBrlZn[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone Set Temperatures.
ActTmpBrlZn[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone Actual Temperatures.
SetTmpBrlZnStb[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone Standby Set Temperatures.
SetTmpBrlZnHdev[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone High Deviation Setpoint. Deviation setpoints are relative to the SetTmpBrlZn or SetTmpBrlZnStb setpoints.
SetTmpBrlZnLdev[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone Low Deviation Setpoint. Deviation setpoints are relative to the SetTmpBrlZn or SetTmpBrlZnStb setpoints.
SetTmpBrlZnHlmt[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone High Limit Setpoint. Limit setpoints are absolute temperature values.
SetTmpBrlZnLlmt[InjUnit, Zone]	No	NUMERIC	Celsius	Barrel Temperature Zone Low Limit Setpoint. Limit setpoints are absolute temperature values.

Table , Barrel Temperature Tokens

Token	Req	Format	Units	Notes
SetDescMldZn[Zone].	No	VSTRING(256)	n/a	Mold Zone Description / Name. Maximum number of zones specified by machine. Single character field for each injection zone. '0' indicates mold zone is OFF, '1' indicates mold zone is ON.
SetCfgMldZn[Zone]	No	CHAR(1)	n/a	Mold Temperature Zone Configuration. Single character field for each mold zone as follows: O: Mold Zone is OFF 0: Mold Zone is Not Supported A: Mold Zone is in AUTO mode T: Mold Zone is in TUNING mode S: Mold Zone is in STANDBY mode M: Mold Zone is in MANUAL mode
SetTmpMldZn[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone Set Temperatures.
ActTmpMldZn[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone Actual Temperatures.
SetTmpMldZnStb[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone Standby Set Temperatures.
SetTmpMldZnHdev[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone High Deviation Setpoint. Deviation setpoints are relative to the SetTmpMldZn or SetTmpMldZnStb setpoints.
SetTmpMldZnLdev[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone Low Deviation Setpoint. Deviation setpoints are relative to the SetTmpMldZn or SetTmpMldZnStb setpoints.
SetTmpMldZnHlmt[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone High Limit Setpoint. Limit setpoints are absolute temperature values.
SetTmpMldZnLlmt[Zone]	No	NUMERIC	Celsius	Mold Temperature Zone Low Limit Setpoint. Limit setpoints are absolute temperature values.

Table , Mould Temperature Tokens

Token	Req	Format	Units	Notes
ActTmpOil	No	NUMERIC	Celsius	Oil Actual Temperature

Token	Req	Format	Units	Notes
SetTmpOil	No	NUMERIC	Celsius	Oil Set Temperature
ActTmpWtrIn	No	NUMERIC	Celsius	Water Intake Actual Temperature
ActTmpWtrOut	No	NUMERIC	Celsius	Water Outlet Actual Temperature
ActTmpCab	No	NUMERIC	Celsius	Cabinet Actual Temperature
ActTmpMlt	No	NUMERIC	Celsius	Melt Actual Temperature

Table , Miscellaneous Temperature Tokens

Token	Req	Format	Units	Notes
ActTimFill[InjUnit]	Yes	NUMERIC	Seconds	Actual Fill Time for each injection unit.
ActTimPlst[InjUnit]	Yes	NUMERIC	Seconds	Actual Plasticization Time for each injection unit.
SetTimCyc	Yes	NUMERIC	Seconds	Overall Cycle Time Setpoint.
ActTimCyc	Yes	NUMERIC	Seconds	Actual Cycle Time.
ActStrCsh[InjUnit]	No	NUMERIC	mm	Actual Stroke Position at Cushion for each injection unit.
ActVolCsh[InjUnit]	No	NUMERIC	cm^3	Actual Volume at Cushion for each injection unit.
ActStrPlst[InjUnit]	No	NUMERIC	mm	Actual Plasticization Stroke for each injection unit.
SetStrPlst[InjUnit]	No	NUMERIC	mm	Plasticization Stroke Setpoint for each injection unit.
ActVolPlst[InjUnit]	No	NUMERIC	cm^3	Actual Plasticization Volume for each injection unit.
SetVolPlst[InjUnit]	No	NUMERIC	cm^3	Plasticization Volume Setpoint for each injection unit.
ActDiaScr[InjUnit]	No	NUMERIC	mm	Actual Screw Diameter for each injection unit.
SetDiaScr[InjUnit]	No	NUMERIC	mm	Screw Diameter Setpoint for each injection unit.
ActStrDcmpPre[InjUnit]	No	NUMERIC	mm	Actual Decompression Stroke Before Plasticization for each injection unit.
SetStrDcmpPre[InjUnit]	No	NUMERIC	mm	Decompression Stroke Before Plasticization Setpoint for each injection unit.
ActVolDcmpPre[InjUnit]	No	NUMERIC	cm^3	Actual Decompression Volume Before Plasticization for each injection unit.
SetVolDcmpPre[InjUnit]	No	NUMERIC	cm^3	Decompression Volume Before Plasticization Setpoint for each injection unit.
ActStrDcmpPst[InjUnit]	No	NUMERIC	mm	Actual Decompression Stroke After Plasticization for each injection unit.
SetStrDcmpPst[InjUnit]	No	NUMERIC	mm	Decompression Stroke After Plasticization Setpoint for each injection unit.
ActVolDcmpPst[InjUnit]	No	NUMERIC	cm^3	Actual Decompression Volume After Plasticization for each injection unit.
SetVolDcmpPst[InjUnit]	No	NUMERIC	cm^3	Decompression Volume After Plasticization Setpoint for each injection unit.
ActStrXfr[InjUnit]	No	NUMERIC	mm	Transfer Stroke Actual for each injection unit.
SetStrXfr[InjUnit]	No	NUMERIC	mm	Transfer Stroke Setpoint for each injection unit.
ActVolXfr[InjUnit]	No	NUMERIC	cm^3	Transfer Volume Actual for each injection unit.
SetVolXfr[InjUnit]	No	NUMERIC	cm^3	Transfer Volume Setpoint for each injection unit.
ActPrsXfrHyd[InjUnit]	No	NUMERIC	bar	Hydraulic Pressure at Transfer Actual for each injection unit.
SetPrsXfrHyd[InjUnit]	No	NUMERIC	bar	Hydraulic Pressure at Transfer Setpoint for each injection unit.
ActPrsXfrCav[InjUnit]	No	NUMERIC	bar	Cavity Pressure at Transfer Actual for each injection unit.
SetPrsXfrCav[InjUnit]	No	NUMERIC	bar	Cavity Pressure at Transfer Setpoint for each injection unit.
ActPrsXfrSpec[InjUnit]	No	NUMERIC	bar	Specific Pressure at Transfer Actual for each injection unit.
SetPrsXfrSpec[InjUnit]	No	NUMERIC	bar	Specific Pressure at Transfer Setpoint for each injection unit.
ActTimXfr[InjUnit]	No	NUMERIC	seconds	Time of Transfer Actual (relative from start of cycle)
SetTimXfr[InjUnit]	No	NUMERIC	seconds	Time of Transfer Setpoint (relative from start of cycle)
ActPrsCavMax	No	NUMERIC	bar	Cavity Maximum Pressure Actual
ActPrsMachHydMax	No	NUMERIC	bar	Machine Hydraulic Pressure Actual Maximum during cycle.
ActPrsMachSpecMax	No	NUMERIC	bar	Machine Specific Pressure Actual Maximum during cycle.
ActSpdPlstMax[InjUnit]	No	NUMERIC	RPM	Plasticization Speed Actual Maximum for each injection unit.
ActSpdPlstAve[InjUnit]	No	NUMERIC	RPM	Plasticization Speed Actual Average for each injection unit.
ActVelPlstMax[InjUnit]	No	NUMERIC	mm/sec	Plasticization Velocity Actual Maximum for each injection unit.
ActVelPlstAve[InjUnit]	No	NUMERIC	mm/sec	Plasticization Velocity Actual Average for each injection unit.
ActFrcClp	No	NUMERIC	KN	Clamp Force Actual
SetFrcClp	No	NUMERIC	KN	Clamp Force Setpoint
ActPrsHldHydMax	No	NUMERIC	bar	Hold Hydraulic Pressure Actual Maximum
ActPrsHldHydAveMax	No	NUMERIC	bar	Hold Hydraulic Pressure Actual Average
ActPrsHldSpecMax	No	NUMERIC	bar	Hold Specific Pressure Actual Maximum
ActPrsHldSpecAveMax	No	NUMERIC	bar	Hold Specific Pressure Actual Average
ActPrsPlstHydMax	No	NUMERIC	bar	Plasticization Hydraulic Pressure Actual Maximum
ActPrsPlstHydAveMax	No	NUMERIC	bar	Plasticization Hydraulic Pressure Actual Average
ActPrsPlstSpecMax	No	NUMERIC	bar	Plasticization Specific Pressure Actual Maximum
ActPrsPlstSpecAveMax	No	NUMERIC	bar	Plasticization Specific Pressure Actual Average

Table , Process Monitoring Parameter Tokens

3.12. Token Summary for other devices

This section is open for the development of new command token sets for non-IMM devices (such as extruders, etc.).

3.13. Unit Summary

In Europe the SI system was defined in 1948 and has been in use since the 1960's (SI = Système International d'Unitiés - International Unit System). It compromises seven basic units and several other units derived from these basic units.

The basic units are:

m	meter
kg	kilogram
s	second
A	Ampere
K	Kelvin
mol	matter quantity (used in chemistry)
cd	Candela

For historical reasons some units that are not recommended for use by the SI system are still in common use in industry (e.g. cm^3 = cubic centimeter, °C = degree Celsius).
In the plastic machinery industry a set of preferred units gained acceptance for the physical quantities used in injection molding machines. These are marked with an asterisk. Of Course these preferred units can vary from one manufacturer to another, since these units are not standardized.

Counters		pure number, no unit
Displacement	*	mm (millimeter)
Dimensions		m (meter)
Energy		J (Joule = Ws)
	*	kWh (kilowatt hour)
Force		N (Newton)
	*	kN (kilonewton)
Mass	*	g (gramme)
		kg (kilogramme)
Mass stream	*	g/s
		kg/s
Power		W (Watt)
	*	kW (kilowatt)
Pressure	*	bar (= Newton per cm^2)
		Pa (Pascal = N/m^2, unicommon)
		hPa (hectopascal, equivalent to mbar
Rotation speed	*	1/min (revolutions per minute)
Temperature	*	degree Celsius (degree centigrade)
Time	*	s (second)
		ms (millisecond)
Velocity	*	mm/s (millimeter per second)
		m/s (meter per second)
		injection speed can also be expressed in cm^3/s
Volume	*	cm^3 (cubic centimeter, not a recommended unit, but widely used
		l (liter)
		ml (milliliter, same as cm^3)
Volume stream	*	cm^3/s

Table , Physical Units Used in Injection Moulding Machines
Many parameters may also be expressed as a unit-less percentage of a maximum value.

3.14. Application Examples

This chapter contains a set of application examples in order to assist in the understanding the commands of the file based date interchange. Keywords are written in capital letters.

3.14.1. Recording of Process Log

JOB file (`PD.JOB`)

```
JOB pd RESPONSE “\\server1\vol1\mach_10\pd.log”;

REPORT spc
  APPEND “\\server1\vol1\mach_10\prot\spc.dat”
  START IMMEDIATE       // start the report immediate
  STOP NEVER            // no special stop condition
  CYCLIC SHOT 1         // record parameters every shot
  SESSION 1000          // record 1000 cycles (=1000
shots)
  PARAMETERS
    DATE,
    TIME,
    COUNT,
    ActCntCyc,
    ActCntCycRej,
    ActFrcClp,
    @ActMyPara;
```

Presentation Layer Response File (`PD.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971207 10:15:32
COMMAND 2 PROCESSED “REPORT command“ 19971207 10:15:32
```

Application Layer Response File

Report response file (`SPC.DAT`)

```
DATE,TIME,COUNT,ActCntCyc,ActCntCycRej,ActFrcClp,@ActMyPa
ra
19971208,10:15:50,1,1000,5,800.4,30.6
19971208,10:16:10,2,1001,5,800.3,30.2
19971208,10:16:30,3,1002,6,799.9,31.0
19971208,10:16:50,4,1003,6,800.0,30.0
19971208,10:17:10,5,1004,6,800.1,30.1
....

19971208,16:10:10,999,1990,8,800.1,30.7
19971208,16:10:30,1000,1999,8,800.7,30.2
```

3.14.2. Recording of Alarm Log

JOB file (ALR.JOB)

```
JOB alr RESPONSE “\\server1\vol1\mach_10\alr.log”;

EVENT myEventAlarms ALARMS REWRITE
“\\server1\vol1\mach_10\prot\alr.dat”
START IMMEDIATE               // start the report
immediate
STOP TIME>= 18:00:00;         // stop recording at 6 pm
```

Presentation Layer Response File (ALR.LOG)

```
COMMAND 1 PROCESSED “JOB command“ 19971207 10:15:32
COMMAND 2 PROCESSED “EVENT command“ 19971207 10:15:33
```

Application Layer Response File

Event response file (ALR.DAT)

```
1,199701208,10:16:30,1002,1,0003,“Value out of range“
2,199701208,10:16:39,1002,0,0003,“Value out of range“
.....
```

3.14.3. Recording of Process Log for a Status View

JOB file (STATUS.JOB)

```
JOB machstat RESPONSE
“\\server1\vol1\mach_10\status.log”;

REPORT status REWRITE
  “\\server1\vol1\mach_10\prot\status.dat” //cyclic
                                          //overwriting
  START IMMEDIATE       // start the report immediate
  STOP NEVER            // no stop condition
  CYCLIC TIME 00:00:10  // record parameters every 10 sec
  PARAMETERS
    ActStsMach;
```

Application Layer Response Files

Report response file (STATUS.DAT)

```
ActStsMach
0A001
```

3.14.4. Recording of Alarm Log for a Status View

JOB file (`ALARM.JOB`)

```
JOB alarm RESPONSE “\\server1\vol1\mach_10\alarm.log”;

EVENT CURRENT_ALARMS REWRITE
“\\server1\vol1\mach_10\prot\alarm.dat”
  START IMMEDIATE       // start the report immediate
  STOP NEVER;           // no stop condition
```

Presentation Layer Response file (`ALARM.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971207 11:35:13
COMMAND 2 PROCESSED “EVENT command“ 19971207 11:35:13
```

Event response file (`ALARM.DAT`)

```
1,199701208,11:50:31,1103,1,0003,“Value out of range“
2,199701208,11:50:31,1103,1,0010,“Clamping force too
high“
```

3.14.5. Download a Data Set (Example 1)

JOB file (`DOWNLOAD.JOB`)

```
JOB download RESPONSE
“\\server1\vol1\mach_10\download.log”;

DOWNLOAD “\\server1\vol1\mach_10\data\dataset_1”
  ACTIVE                // overwrite active data set
  START IMMEDIATE;      // start the download immediate
                        // after writing this file
```

Presentation Layer Response file (`DOWNLOAD.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971207 11:35:13
COMMAND 2 ERROR 06 00000016 “DOWNLOAD operation denied.“
19971207 11:35:13
```

Application Layer Response Files

none

3.14.6. Download a Data Set (Example 2)

JOB file (`DOWNLOAD.JOB`)

```
JOB download RESPONSE
“\\server1\vol1\mach_10\download.log”;

DOWNLOAD “\\server1\vol1\mach_10\data\dataset_2”
  dataset_2                // overwrite archives
dataset_2
  START SetDescPrt = “A1000140“; // start the download if
                                 // part name = A1000140
```

Presentation Layer Response file (`DOWNLOAD.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971207 14:15:43
COMMAND 2 PROCESSED “DOWNLOAD command“ 19971207 14:15:43
```

Application Layer Response Files

none

3.14.7. Setting Several Parameters

JOB file (`SET.JOB`)

```
JOB set RESPONSE “\\server1\vol1\mach_10\set.log”;
SET SetDescMold “MOLD 1314“;
SET SetTimMach 10150019971201;
SET SetCntCyc 1000;
SET @SetMyPara_1 23.6;
SET @SetMyPara_2 100000.6;
```

Presentation Layer Response file (`SET.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971201 10:15:32
COMMAND 2 PROCESSED “SET command“ 19971201 10:15:32
COMMAND 3 PROCESSED “SET command“ 19971201 10:15:32
COMMAND 4 PROCESSED “SET command“ 19971201 10:15:33
COMMAND 5 PROCESSED “SET command“ 19971201 10:15:33
COMMAND 6 ERROR 06 00000021 “SET value out of range“
19971201 11:15:33
```

Application Layer Response Files

none

3.14.8. GETID Command

JOB file (`GETID.JOB`)

```
JOB set RESPONSE “\\server1\vol1\mach_10\getid.log”;
GETID “\\server1\vol1\mach_10\config.dat”;
```

Presentation Layer Response file (`GETID.LOG`)

```
COMMAND 1 PROCESSED “JOB command“ 19971201 10:15:32
COMMAND 2 PROCESSED “GETID command“ 19971201 10:15:32
```

Application Layer Response File (CONFIG.DAT)

```
ActTimFill[1],N,3,1,0,“Seconds“,“Actual Fill Time for
injection unit 1“
ActTimFill[2],N,3,1,0,“Seconds“,“Actual Fill Time for
injection unit 2“
ActTimPlst[1],N,3,1,0,“Seconds“,“Actual Plasticization
for injection unit 1“
ActTimPlst[2],N,3,1,0,“Seconds“,“Actual Plasticization
for injection unit 2“
SetTimCyc,N,3,1,1,“Seconds“,“Overall Cycle Time Setpoint“
ActTimCyc,N,3,1,0,“Seconds“,“Actual Cycle Time“
...

@SetMyTextPara_1,A,20,0,1,““,“First Text parameter“
@SetMyTextPara_2,A,20,0,1,““,“Second Text parameter“
@SetMyBoolPara_1,B,1,0,1,““,“First boolean parameter“
@SetMyBoolPara_2,B,1,0,1,““,“Second boolean parameter“
@ActMyNumericPara_1,N,4,1,0,“mm“,“First numeric
parameter“
@SetMyNumericPara_2,N,3,2,1,“kN“,“Second numeric
parameter“
@20100,N,4,1,1,“bar“,“Third numeric parameter“
@100,N,8,0,1,““,“Another numeric parameter“
...
```