Server Communication ICD

SPELL version 2.4

Distribution list

Full Report:
SES-SSO-SOE

For Information Purposes:
Open Source

Reference:
SES-SSO-SOE-SPELL-2013/01

Date issued:
July 2013

Index of Content

1 INTRODUCTION

1.1 Purpose of this document

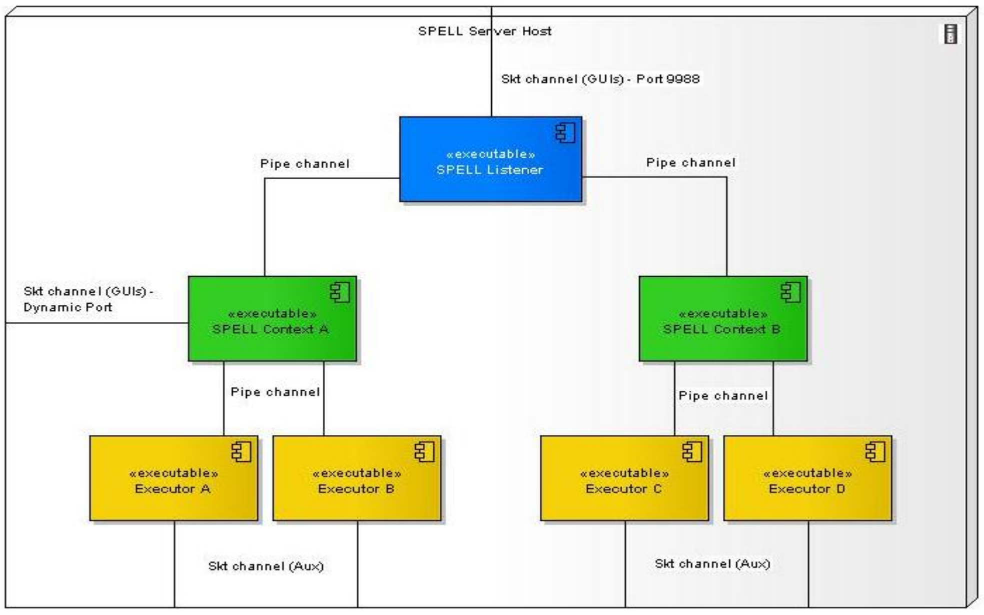
This document describes the mechanisms available to communicate with a SPELL execution environment, SPELL server for short, from any external software component. API libraries are available for Java and C++. The usage of these libraries and the message protocol between the client and the SPELL server is detailed in the next sections.

2 SYSTEM ARCHITECTURE

2.1 Client-Server

The SPELL execution environment is based on a client-server architecture. The client component is normally the SPELL-GUI application, which communicates via TCP/IP with processes on the SPELL server side. In this document, the mechanisms and protocols used by the SPELL-GUI are described, so that third party components that are able to interact with the SPELL server can be easily developed.

The process structure of the SPELL server side is depicted in the image below:



The first entry point to the execution environment is the SPELL Listener (in blue). This process is so to say the access portal to the environment and it is the first component to which the client applications need to connect.

Once a client is connected to a listener, it can retrieve information about the available contexts (i.e. spacecraft environments) available on the SPELL server. There is one Context process controlled spacecraft. Via the Listener, the client application can start, stop or kill available context processes.

When an appropriate Context is chosen, the client application needs to request connection to that Context process to the Listener. Then a secondary TCP/IP connection between the client and the Context process is established. From this point on, the Listener process plays little role during the execution of procedures, and messages between the client and the Listener will be exchanged only to report changes in Context processes status and to perform control operations on the contexts.

Finally, when a client application opens a procedure for execution, a new process is created under the control of the Context: the Executor process. There is one executor per procedure active on the server.

The Executor process is the most important one in the SPELL server and it is in charge of performing the actual execution of the procedure directives; it also loads the SPELL Driver which is configured for the context. This means that it is the Executor, and only the Executor, the process that establishes connections to the Ground Control systems.

The commands to control the procedures (RUN, STEP, ABORT, etc.) sent by the client application are handled by the Context, and forwarded to the appropriate Executor. Note that there may be many Executors active at the same time under the control of one Context process; therefore the Context is acting as a message dispatcher.

In a similar way, all the notification messages (items, status, current line, etc.) sent by the Executor are handled by the context and forwarded to the appropriate client applications. Note also that there may be many client applications (SPELL-GUI and others) connected to the Context at any moment, and also attached to one or more Executors within that context. All clients attached -associated- to an Executor shall receive all the feedback messages generated by the Executor.

2.2 Operations on the listener

Once a client application is connected with the SPELL listener, the following operations can be performed:

- Get list of available S/C Contexts
- Get the status of a given Context process
- Start, stop or kill a context
- Attach to or detach from a context

Note that "attaching" and "detaching" to and from a Context means establishing or terminating the TCP/IP connection between a client application and the Context process. The message protocol to perform these operations is described in sections 4 and 5.

2.3 Operations on the context

Once a client application is attached to a given Context process, the following operations can be performed:

- Retrieve the list of active executors (procedures active in the context)
- Retrieve the list of available procedures
- Retrieve procedure properties, source code and static information
- Retrieve server files (log files, ASRUN files, data dictionaries, others)
- Open, close or kill executors (procedures)
- Take/release control of procedures and start/stop monitoring sessions
- Put procedures in background mode or take them back to foreground (take control back)
- Get and receive information about active procedures (configuration and status)
- Replay procedure executions (replay of the ASRUN files)

The message protocol to perform these operations is described in chapter 6 and 7.

2.4 Operations with procedures

When a client application is the controlling client of a procedure, an additional number of control operations can be performed. A client application is the controlling client of a procedure in the following cases:

1. The client application requested opening the procedure. In this case the client application is the controlling client automatically and from the moment the procedure is loaded.

2. The client application invoked a Take Control operation on the procedure.

The operations available for a controlling client are the following:

- Issue procedure commands (RUN, PAUSE, ABORT, STEP, etc.)
- Provide answer to Prompt requests (procedure input)
- Process feedback notifications generated by the procedure execution
- Inspect and modify the data variables on the procedure execution scope
- Inspect and modify SPELL dictionaries on the procedure execution scope
- Execute arbitrary SPELL statements on the procedure execution scope
- Inspect and modify the procedure execution configuration (execution delay, by-step, etc.)
- Retrieve the executor runtime information (status, current line, associated clients, etc.)

The message protocol to perform these operations is described in sections 8 and 9.

2.5 Communication APIs

Several APIs are provided as open source packages, ready to be used by third-party applications in order to communicate with a SPELL execution environment:

- Eclipse RCP plugins (no dependencies to the graphical framework)
- Pure Java jar files
- C++ dynamic libraries and headers.

The RCP API reuses internal models and mechanisms used by the standard SPELL-GUI, therefore the client applications using these plugins do not need to re-work the low level operations and IPC protocol mechanisms.

The other two APIs provide lower level interfaces and the client applications are in charge of creating and processing IPC messages at a lower level.

3 COMMUNICATION PROTOCOL BASICS

3.1 Connecting to server interfaces

The first operation to be carried out once the TCP socket connection is established, is to exchange a TCP key identifier (an integer) with the server. This client key is used to identify the sender and receiver of the protocol messages, and therefore to dispatch them properly. The TCP client key is an integer (two bytes).

The server will assign client keys sequentially to all clients connecting, unless the client suggest one (and obviously the suggested key is available). This particular key exchange is meant to give re-connection capabilities using with the same client key after a possible contingency.

Therefore the first steps are:

1. If the client application should re-use a client key, send the key integer to the server as the first operation (two bytes)
2. Otherwise, send a zero-key (two zero bytes) to the server as the first operation.
3. Wait and read two bytes from the server to compose the assigned client key.

A typical Java code for reading the key follows:

```
DataInputStream in = …;
byte keyb1 = in.readByte();
byte keyb2 = in.readByte();
int key = keyb1;
int myKey = key << 8;
myKey += keyb2;
```

3.2 Disconnecting from server interfaces

In order to properly disconnect from a SPELL server interface, the client application should send an "EOC" (End-Of-Conversation) message. This is just a SPELL message with a special type. Once the EOC message is sent, the client can shut the socket connection down normally. Please refer to the following sections regarding the SPELL messages structure.

The EOC message type identifier is "eoc". The parameters of this message follows:

- **Type:** "eoc"
- **Sender:** "CLT"

Other parameters like a message identifier (id) are not needed.

3.3 SPELL messages

The SPELL messages are composed of key-value pairs, or properties. The amount of properties depend on the particular message, but at a minimum the message will contain the following elements (exceptions to this rule are explained case by case):

- **Type:** defines the nature of the message. For example, this property is used to distinguish synchronous requests from asynchronous messages.
- **Sender:** identififer of the component that originated the message. The sender is not necessarily the same SW component that owns the IPC communication interface (i.e. that holds the client IPC key).
- **Receiver:** analogous to the Sender property, identifies the SW component that should receive the message. Again, the receiver is not necessary the component that owns the IPC communication interface on the other side.
- **Id:** identifier of the message. Allows distinguishing messages of the same type but different purpose.
- **IpcKey:** holds the key identifying the IPC interface originating the message.

3.4 SPELL messages encoding

The messages are encoded as streams of bytes. The entire message length is encoded first in the 4 first bytes of the stream. Then, for each key-value pair, the length of the key and the length of the value are encoded sequentially in the stream. The algorithm follows:

1. Iterate over all message properties
2. For each property, calculate the length of the key string, and the length of the value string.
3. Note that the properties shall include at a minimum the ones described in previous section.
4. Write a character "\1" in a memory buffer of bytes. This is a flag used for message compression support, but it is not needed at the moment for client applications.
5. Write the property bytes in the buffer in this order:
    a. 2 bytes for the length of the key string
    b. Bytes of the key string
    c. 2 bytes for the length of the value string
    d. Bytes of the value string
6. Repeat until all the bytes corresponding to all message properties have been written.
7. Calculate the resulting total length of the byte stream
8. Write the total message stream length through the socket.
9. Write the contents of the memory buffer through the socket.

A typical Java code for writing the messages follows.

```
    ByteArrayOutputStream buf = new ByteArrayOutputStream();
    buf.write('\1');
    for( String key : msg.getKeys() )
    {
        String value = msg.get(key);
        buf.write( encodeLength2( key.lenght() ), 0, 2 );
        buf.write( key.getBytes() );
        buf.write( encodeLength2( value.lenght() ), 0, 2 );
        buf.write( value.getBytes() );
    }

    byte[] encodedLength = encodeLength4( buf.size() );
    for(byte b : encodedLength) sktDataOutputStream.writeByte(b);

    for(byte b : buf.toByteArray() )
    {
        sktDataOutputStream.writeByte(b);
    }

    sktDataOutputStream.flush();

…
private byte[] encodeLength4( int length )
{
    byte[] bytes = new byte[4];
    for( int i=3; i>=0; i--)
    {
        bytes[i] = (byte) (length & 0xff);
        length >>= 8;
    }
    return bytes;
}

private byte[] encodeLength2( int length )
{
    byte[] bytes = new byte[2];
    bytes[1] = (byte) (length & 0xff);
    length >>= 8;
    bytes[0] = (byte) (length & 0xff);
    return bytes;
}
```

3.5 SPELL messages decoding

The client application shall read first the total message stream length (4 first bytes of the message stream). The algorithm follows:

1. Read the first 4 bytes and find out the total message length.
2. Read the complete stream and store it on a memory buffer.
3. If the first byte of the data stream is '\1', the message is not compressed. If the first byte is '\2' the message is compressed with GZIP.
4. If the message is compressed, use a GZIP API to uncompress the data stream.
5. Iterate over the data stream to decode the key-value pairs, knowing that the length of the keys and values are available sequentially.

A typical Java code (no error handling) for reading the messages follows.

```
int length = 0;
short b1, b2;
String key = "";
String value = "";

for( int pos = 0 ; pos < data.length; )
{
      b1=(data[pos]>= 0) ? data[pos] : (short) (data[pos] + 256);
      b2=(data[pos+1] >= 0) ? data[pos+1] : (short) (data[pos+1] + 256);
      length = b1 * 256 + b2;
      pos += 2;

      key = new String( data,  pos, length );
      pos += length;

      b1=(data[pos] >= 0) ? data[pos] : (short) (data[pos] + 256);
      b2=(data[pos+1] >= 0) ? data[pos+1] : (short) (data[pos+1] + 256);
      length = b1 * 256 + b2;
      pos += 2;

      value = new String( data, pos, length );
      pos += length;
      propertiesMap.put(key,value);
}
```

A code example for uncompressing messages in Java follows:

```java
ByteArrayInputStream bis = new ByteArrayInputStream(data);
ByteArrayOutputStream bos = new ByteArrayOutputStream();
GZIPInputStream gis = new GZIPInputStream(bis);
int numRead = 0;
byte[] output = new byte[512];
while(numRead != -1)
{
      numRead = gis.read(output, 0, output.length);
      if (numRead >0)
      {
            bos.write(output,0,numRead);
      }
}
bis.close();
bos.close();
```

3.6 SPELL message types

There is a variety of SPELL messages that will be described in the following. Nevertheless, some main types are to be described first:

1. One-way messages: these are sent asynchronously and the sender does not expect any response for such messages. The calls to send these messages shall be non-blocking.

2. Request messages: these messages are sent synchronously and the sender shall block and wait for a response.

3. Response messages: associated to requests.

4. Error messages: when a client performs a request, an error message can be returned instead of a regular response. This error message will provide information about why the request could not be processed (or failed) on the server side.

4 MESSAGES SENT TO LISTENER

4.1 Generic fields

All messages sent to the listener will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	CLT
Receiver	LST
IpcKey	<client IPC key>

All request responses from the listener will arrive with at least the following fields:

FIELD	CONTENTS
Sender	LST
Receiver	CLT
IpcKey	<listener IPC key>
Type	response

In case of failures, instead of receiving a SPELL response as the answer to a blocking request, the listener may send a SPELL error message. The error messages will contain at least the following fields:

FIELD	CONTENTS
Sender	LST
Receiver	CLT
IpcKey	<listener IPC key>
Type	error
ErrorMsg	Error message
ErrorReason	Reason of the error
FatalError	True or False. True if the failure caused a major problem on the server side and some recovery actions -maybe manual restart- are needed.

4.2 Login

Once the IPC connection with the SPELL listener is established and the client application has an IPC key available, the client must send a login request.

Request

FIELD	CONTENTS
Id	REQ_GUI_LOGIN
Type	request
Host	<client host name>

Response

FIELD	CONTENTS
Id	RSP_GUI_LOGIN
Type	response

4.3 Get list of available contexts

Request used in order to get the list of available Contexts defined on the SPELL server.

Request

FIELD	CONTENTS
Id	REQ_CTX_LIST
Type	request

Response

FIELD	CONTENTS
Id	RSP_CTX_LIST
Type	response
ContextList	Comma-separated list of context names

4.4 Get context information

Request used in order to get information about a given SPELL context.

Request

FIELD	CONTENTS
Id	REQ_CTX_INFO
Type	request
ContextName	Name of the context

Response

FIELD	CONTENTS
Id	RSP_CTX_LIST
Type	response
ContextStatus	One of the following: AVAILABLE, STARTING, RUNNING, ERROR, KILLED UNKNOWN
ContextPort	IPC port where the context is listening for client applications
ContextDriver	Name of the SPELL driver associated to this context
ContextDescription	Description string of the context
ContextSC	Name of the spacecraft associated to the context
ContextGCS	Name of the ground control system associated to the context
ContextFamily	Free identifier for the family or category of the ground control system instance associated to the context (e.g. PRIMARY, BACKUP, etc.)
MaxProc	Maximum amount of active procedures permitted by the context (0 means no limit)

4.5 Start a context process on the server

Request used in order to start a given SPELL context on the execution environment.

Request

FIELD	CONTENTS
Id	REQ_OPEN_CTX
Type	request
ContextName	Name of the context

Response

FIELD	CONTENTS
Id	RSP_OPEN_CTX
Type	response

4.6 Stop a context process on the server

Request used in order to stop a given SPELL context on the execution environment.

Context stop request

FIELD	CONTENTS
Id	REQ_CLOSE_CTX
Type	request
ContextName	Name of the context

Response

FIELD	CONTENTS
Id	RSP_CLOSE_CTX
Type	response

4.7 Destroy a context process on the server

Request used in order to hard-kill a given SPELL context on the execution environment.

Context destroy request

FIELD	CONTENTS
Id	REQ_DESTROY_CTX
Type	request
ContextName	Name of the context

Response

FIELD	CONTENTS
Id	RSP_DESTROY_CTX
Type	response

4.8 Attach client to a context process on the server

Request used in order to attach the client application to a given SPELL context on the execution environment.

Context attach request

FIELD	CONTENTS
Id	REQ_ATTACH_CTX
Type	request
ContextName	Name of the context

Response

FIELD	CONTENTS
Id	RSP_ATTACH_CTX
Type	response
<etc>	Same fields as for REQ_CTX_INFO

Once the response to the attach message is received, if the information given by the server is correct, the client application shall open a new IPC communication interface and connect it to the port indicated in ContextPort field.

A login request shall be then sent again to the Context process (see context protocol). Note that there is no DETACH request in the listener protocol: detaching from a context is a message exchange with the Context process itself, and the listener does not play a role at the moment.

4.9 Logout

Request used to finish a connection to the listener process.

Logout request

FIELD	CONTENTS
Id	REQ_GUI_LOGOUT
Type	request
Host	<client host name>

Response

FIELD	CONTENTS
Id	RSP_GUI_LOGOUT
Type	response

5 MESSAGES COMING FROM LISTENER

5.1 Generic fields

All messages sent from the listener will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	LST
Receiver	CLT
IpcKey	<listener IPC key>

All request responses and one-way messages sent to the listener shall have at least the following fields:

FIELD	CONTENTS
Sender	CLT
Receiver	LST
IpcKey	<client IPC key>
Type	response or oneway

The client applications are not meant to send error responses to the listener at the moment (they would be ignored).

5.2 Notification of context operations

Issued by the listener when the status of a context process changes.

Notification message

FIELD	CONTENTS
Id	MSG_CONTEXT_OP
Type	oneway
<etc>	Same messages as for Context Information Request

6 MESSAGES SENT TO CONTEXT

6.1 Generic fields

All messages sent to the context will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	CLT
Receiver	CTX
IpcKey	<client IPC key>

All request responses from the context will arrive with at least the following fields:

FIELD	CONTENTS
Sender	CTX
Receiver	CLT
IpcKey	<context IPC key>
Type	response

In case of failures, instead of receiving a SPELL response as the answer to a blocking request, the context may send a SPELL error message. The error messages will contain at least the following fields:

FIELD	CONTENTS
Sender	CTX
Receiver	CLT
IpcKey	<context IPC key>
Type	error
ErrorMsg	Error message
ErrorReason	Reason of the error
FatalError	True or False. True if the failure caused a major problem on the server side and some recovery actions -maybe manual restart- are needed.

6.2 Login

Once the IPC connection with the SPELL context is established and the client application has an IPC key available, the client must send a login request.

Login request

FIELD	CONTENTS
Id	REQ_GUI_LOGIN
Type	request
Host	<client host name>

Login response

FIELD	CONTENTS
Id	RSP_GUI_LOGIN
Type	response

6.3 Close context

Issued to the context in order to make the process stop.

Close message

FIELD	CONTENTS
Id	MSG_CLOSE
Type	oneway

6.4 Obtain the list of available procedures

Used to get the list of existing procedures that can be started on the context.

Procedure list request

FIELD	CONTENTS
Id	REQ_PROC_LIST
Type	request

Response

FIELD	CONTENTS
Id	RSP_PROC_LIST
Type	response
ProcList	List of procedure identifier/name pairs. Separator character for list elements is '\3'. Separator character for identifier/name is "|". Example: procId1|Procedure Name1\3procId2|Procedure Name 2…

6.5 Obtain the list of available executors

Used to get the list of active executors on the context (procedures open)

Executor list request

FIELD	CONTENTS
Id	REQ_EXEC_LIST
Type	request

Response

FIELD	CONTENTS
Id	RSP_EXEC_LIST
Type	response
ExecutorList	List of procedure instance identifiers. Separator character for list elements is '\3'. Example: procId1#0\3procId2#3\3….

6.6 Obtain client information

Used to get information about another client application connected to the context.

Client information request

FIELD	CONTENTS
Id	REQ_CLIENT_INFO
Type	request
GuiKey	IPC key of the other client application

Response

FIELD	CONTENTS
Id	RSP_CLIENT_INFO
Type	response
Host	Host name of the machine where the other client is connecting from
GuiMode	Mode of the client application: CONTROL or MONITOR

6.7 Obtain executor information

Used to get information about an executor (active procedure) running on the Context.

Executor information request

FIELD	CONTENTS
Id	REQ_EXEC_INFO
Type	request
ProcId	Instance identifier of the procedure. Example: procId1#2

Response

FIELD	CONTENTS
Id	RSP_CLIENT_INFO
Type	response
ProcName	Name of the procedure (not to confuse with procedure identifier)

ExecutorStatus	Status of the executor. One of the following: UNINIT, LOADED, RUNNING, WAITING, PAUSED, ERROR, ABORTED, FINISHED, PROMPT, RELOADING, UNKNOWN
Condition	A string describing a condition expression if the procedure is evaluating a SPELL expression before running. Empty in normal cases.
GuiControl	IPC key of the client application controlling the procedure. May be empty. If the value is <BACKGROUND> the procedure is in background mode and there is no controlling GUI.
GuiControlHost	Host of the client application controlling the procedure. May be empty.
GuiList	List of client applications monitoring (not controlling) the procedure. Separator of the list elements is ",". The clients are indicated in the form <IPC key>:<hostname>.
ParentId	Parent procedure instance identifier if any. This field may not be present.
ParentLine	Line of the parent procedure where the call to StartProc took place. This field may not be present.
OpenMode	Describes the executor open mode. A comma-separated list of the following: Automatic:True or Automatic:False Blocking:True or Blocking:False Visible:True or Visible:False
ActionLabel	May not be present. Describes the User Action label if it has been defined.
ActionEnabled	True if a User Action is defined and enabled. May not be present.

6.8 Obtain procedure source code

Used to get the source code of a given procedure.

Procedure code request

FIELD	CONTENTS
Id	REQ_PROC_CODE
Type	request
ProcId	Identifier of the procedure, without instance number. E.g. Main/procId1.
CurrentChunk	May not be present. To be used if the procedure code is too big and it has been splitted in chunks (see response message)

Response

FIELD	CONTENTS
Id	RSP_PROC_CODE
Type	response
ProcId	Identifier of the procedure
CurrentChunk	Current chunk of lines of code given.
TotalChunks	Total number of chunks. If zero, the code is not splitted in chunks.
ProcCode	Source code lines. The line separator is '%C%'

If the first response indicates a non-zero number of TotalChunks, the client application shall repeat the procedure request again until all chunks are received. The CurrentChunk field is used to indicate which chunk should be sent (0-TotalChunks-1). Each response given will have a corresponding CurrentChunk indicator.

6.9 Obtain available procedure instance number

Used to get an available instance number for starting a procedure. There may be several instances of the same procedure running on the server; therefore it is necessary to ask the execution environment to provide a free instance identifier. The procedure identifier plus the instance number is the "instance identifier", which shall be used by the client application to start a procedure.

Instance number request

FIELD	CONTENTS
Id	REQ_INSTANCE_ID
Type	request
ProcId	Identifier of the procedure, without instance number. E.g. Main/procId1.

Response

FIELD	CONTENTS
Id	RSP_INSTANCE_ID
Type	response
ProcId	Identifier of the procedure
InstanceId	Instance identifier to be used.

6.10 Obtain procedure properties

Used static information about a procedure

Procedure properties request

FIELD	CONTENTS
Id	REQ_PROC_PROP
Type	request
ProcId	Identifier of the procedure, without instance number. E.g. Main/procId1.

Response

FIELD	CONTENTS
Id	RSP_PROC_PROP
Type	response
<key>	<value>

All procedure properties (the number and codes is not imposed by SPELL but by the procedure header) are provided as key-value pairs of the SPELL message.

6.11 Start an executor (open a procedure)

Used to start a procedure execution on the context.

Open executor request

FIELD	CONTENTS
Id	REQ_OPEN_EXEC
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0
GuiMode	CONTROL
Arguments	Procedure argument list Separator for list items is ',,' Arguments are given as "arg=value"

Response

FIELD	CONTENTS
Id	RSP_OPEN_EXEC
Type	response

6.12 Close an executor (close a procedure)

Used to close a procedure execution on the context.

Close executor request

FIELD	CONTENTS
Id	REQ_CLOSE_EXEC
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0

Response

FIELD	CONTENTS
Id	RSP_CLOSE_EXEC
Type	response

6.13 Kill an executor (close a procedure)

Used to kill a procedure execution on the context.

Kill executor request

FIELD	CONTENTS
Id	REQ_KILL_EXEC
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0

Response

FIELD	CONTENTS
Id	RSP_KILL_EXEC
Type	response

6.14 Attach to an executor (monitor or take control)

Used to associate the client application to a given executor. Depending on the GUI mode given, the client application will be in control of the executor, or it will start a monitoring session.

Attach executor request

FIELD	CONTENTS
Id	REQ_ATTACH_EXEC
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0
GuiMode	CONTROL or MONITOR

Response

FIELD	CONTENTS
Id	RSP_ATTACH_EXEC
Type	response

6.15 Detach from an executor (release a procedure)

Stop controlling a procedure execution, or finish a monitoring session.

Close executor request

FIELD	CONTENTS
Id	REQ_DETACH_EXEC
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0

Response

FIELD	CONTENTS
Id	RSP_DETACH_EXEC
Type	response

6.16 Put a procedure in background

Stop controlling an executor an let it run in the background, without controlling client.

Background request

FIELD	CONTENTS
Id	REQ_SET_BACKGROUND
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0

Response

FIELD	CONTENTS
Id	RSP_SET_BACKGROUND
Type	response

6.17 Remove control of an executor

Force removing the control of a procedure, disconnecting its controlling client. Should not be used in nominal cases.

Remove control request

FIELD	CONTENTS
Id	REQ_REMOVE_CONTROL
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0

Response

FIELD	CONTENTS
Id	RSP_REMOVE_CONTROL
Type	response

6.18 Obtain path for a standard server file associated to an executor

Used to get the absolute path to a standard SPELL file (ASRUN or Executor Log file) associated to an active procedure.

Path request

FIELD	CONTENTS
Id	REQ_SERVER_FILE_PATH
Type	request
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0
ServerFileId	ASRUN or EXECUTOR_LOG

Response

FIELD	CONTENTS
Id	RSP_SERVER_FILE_PATH
Type	response
FilePath	Absolute path to the file

6.19 Obtain path for directory containing standard server files

Used to get the absolute path to a standard SPELL file (ASRUN or Executor Log file) associated to an active procedure.

Path request

FIELD	CONTENTS
Id	REQ_SERVER_FILE_PATH
Type	request
ServerFileId	ASRUN or EXECUTOR_LOG

Response

FIELD	CONTENTS
Id	RSP_SERVER_FILE_PATH
Type	response
FilePath	Absolute path to the directory

6.20 Obtain list of all available ASRUN files

Used to get a list of all existing ASRUN files for the context.

Path request

FIELD	CONTENTS
Id	REQ_ASRUN_LIST
Type	request

Response

FIELD	CONTENTS
Id	RSP_ASRUN_LIST
Type	response
FileList	List of ASRUN file names (no containing directory). Separator for list items is '\3'.

6.21 Obtain list of data directories on server

Used to get a list of all existing SPELL data directories on the context.

Path request

FIELD	CONTENTS
Id	REQ_LIST_DATADIRS
Type	request

Response

FIELD	CONTENTS
Id	RSP_LIST_DATADIRS
Type	response
FileList	List of directory names (full paths) Separator for list items is '\3'.

6.22 Obtain list of files within a directory on the server

Used to get a list of all existing files within a given directory.

Path request

FIELD	CONTENTS
Id	REQ_LIST_FILES
Type	request

Response

FIELD	CONTENTS
Id	RSP_LIST_FILES
Type	response
FileList	List of file names (no full paths) Separator for list items is '\3'.

7 MESSAGES RECEIVED FROM CONTEXT

7.1 Generic fields

All messages sent by the context will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	CTX
Receiver	CLT
IpcKey	<context IPC key>

All request responses given to the context shall have at least the following fields:

FIELD	CONTENTS
Sender	CLT
Receiver	CTX
IpcKey	<client IPC key>
Type	response

At the present version the system is not prepared to process error messages coming from client applications.

7.2 Notification of executor operation

Notification message reporting newly open procedures, procedures changing status or being killed/closed, etc. Notifications for all executors on the context will be sent, no matter if the client application is attached (controlling or monitoring) those executors.

Executor operation notification

FIELD	CONTENTS
Id	MSG_EXEC_OP
Type	oneway
ProcId	Instance identifier (procId + instance number), e.g. Main/proc1#0
ExecOp	Identifier of the operation: OPEN, CLOSE, KILL, ATTACH, DETACH, STATUS, CRASH, UNKNOWN
GuiKey	May not be present. Indicates the client application performing the executor operation, if any.
GuiMode	May not be present. Mode of the client application performing the operation. In case of ATTACH operation it will indicate the mode in which the client application is attaching: CONTROL or MONITOR.
ExecutorStatus	Indicates the status of the executor. The value may not be relevant in operations like KILL or DETACH.

8 MESSAGES SENT TO PROCEDURES

The communication with procedures (i.e. executor processes) takes place through the Context process, which acts as a dispatcher. Therefore the same IPC interface used to interact with the Context is to be used to interact with procedures running within that context. The difference in the messages is that the message receiver property indicates the procedure instance identifier.

8.1 Generic fields

All messages sent to procedures will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	CLT
Receiver	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
IpcKey	<client IPC key>
ProcId	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0

All request responses sent by the procedure shall have at least the following fields:

FIELD	CONTENTS
Sender	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
Receiver	CLT
IpcKey	<executor IPC key>
Type	response

Error messages may also arrive from the Executor process, having at least the following fields:

FIELD	CONTENTS
Sender	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
Receiver	CLT
IpcKey	<executor IPC key>
Type	error
ErrorMsg	Error message

ErrorReason	Reason of the error
FatalError	True or False. True if the failure caused a major problem on the server side and some recovery actions -maybe manual restart- are needed.

8.2 Execution control commands (only controlling clients)

Commands to control the execution of the procedure.

Command message

FIELD	CONTENTS
Id	CMD_ABORT, CMD_RUN, CMD_ACTION, CMD_GOTO, CMD_PAUSE, CMD_RECOVER, CMD_RELOAD, CMD_SCRIPT, CMD_SKIP, CMD_STEP, CMD_STEP_OVER
Type	oneway
ProcId	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
Text	Valid for CMD_ABORT only. Used to provide an explanatory text for the abort action.
Script	For CMD_SCRIPT only. Contains the SPELL code that needs to be manually executed on the procedure scope.
GotoLabel	For CMD_GOTO only. Contains the target Step label to perform a manual goto.
GotoLine	For CMD_GOTO only. Contains the target line number to perform a manual goto.

8.3 Set executor configuration (only controlling clients)

Set the configuration parameters for an active procedure execution.

Request

FIELD	CONTENTS
Id	REQ_SET_CONFIG
Type	request
RunInto	True/False

ByStep	True/False
BrowsableLib	False (deprecated)
ForceTcConfirm	True/False
ExecDelay	<integer, milliseconds of delay>

Response

FIELD	CONTENTS
Id	RSP_SET_CONFIG
Type	response

8.4 Obtain executor configuration

Obtain the runtime parameters for an active procedure execution.

Request

FIELD	CONTENTS
Id	REQ_GET_CONFIG
Type	request

Response

FIELD	CONTENTS
Id	RSP_SET_CONFIG
Type	response
RunInto	True/False
ByStep	True/False
BrowsableLib	False (deprecated)
ForceTcConfirm	True/False
ExecDelay	<integer, milliseconds of delay>

8.5 Obtain executor configuration

Obtain the configuration parameters for an active procedure execution.

Request

FIELD	CONTENTS
Id	REQ_GET_CONFIG
Type	request

Response

FIELD	CONTENTS
Id	RSP_GET_CONFIG
Type	response
RunInto	True/False
ByStep	True/False
BrowsableLib	False (deprecated)
ForceTcConfirm	True/False
ExecDelay	<integer, milliseconds of delay>

8.6 Send an answer to a prompt (only controlling clients)

Send a prompt answer to a procedure in PROMPT status

Message

FIELD	CONTENTS
Id	MSG_PROMPT_ANSWER
Type	oneway
ReturnValue	Result of the prompt

8.7 Cancel a prompt (only controlling clients)

Cancel a prompt in a procedure in PROMPT status.

Message

FIELD	CONTENTS
Id	MSG_CANCEL
Type	oneway

8.8 Send acknowledge message (only controlling clients)

The ACK message shall be sent by the controlling clients of a procedure as a response of the SPELL procedure data notification messages (see next chapter).

Message

FIELD	CONTENTS
Id	ACKNOWLEDGE
Type	oneway
Sequence	Sequence number of the notification being acknowledged.

8.9 Toggle a breakpoint (only controlling clients)

Set or unset a breakpoint on a procedure.

Request

FIELD	CONTENTS
Id	REQ_SET_BREAKPOINT
Type	request
BreakpointProc	Identifier of the code where the breakpoint needs to be set/unset
BreakpointLine	Line of the source code where the breakpoint needs to be set/unset
BreakpointType	PERMANENT, TEMPORARY

Response

FIELD	CONTENTS
Id	RSP_GET_BREAKPOINT
Type	response

8.10 Clear all breakpoints (only controlling clients)

Clear all breakpoints set on a procedure.

Request

FIELD	CONTENTS
Id	REQ_CLEAR_BREAKPOINT
Type	request
BreakpointProc	Identifier of the code where the breakpoint needs to be set/unset
BreakpointLine	Line of the source code where the breakpoint needs to be set/unset
BreakpointType	PERMANENT, TEMPORARY

Response

FIELD	CONTENTS
Id	RSP_CLEAR_BREAKPOINT
Type	response

8.11 Retrieve the contents of a SPELL Data Container

Obtain the contents of a SPELL data container (IVARS, ARGS) of a procedure.

Message

FIELD	CONTENTS
Id	REQ_GET_DICTIONARY
Type	request
DictName	ARGS or IVARS
CurrentChunk	Used only when getting a dictionary divided in chunks. Indicates the next chunk to be retrieved.

Response

FIELD	CONTENTS
Id	RSP_GET_DICTIONARY
Type	response
DictContents	Contents of the dictionary or chunk in the format: <variable name>\5<variable value>\5<variable format>\5<confirm get>
CurrentChunk	Used if the dictionary is divided in chunks. Indicates the chunk given.
TotalChunks	Used if the dictionary is divided in chunks. Indicates the total amount of chunks available.

8.12 Update the contents of a SPELL Data Container (controlling clients only)

Update the contents of a SPELL data container (IVARS, ARGS) of a procedure.

Message

FIELD	CONTENTS
Id	REQ_UPD_DICTIONARY
Type	request
DictName	ARGS or IVARS
DictMergeNew	true/false
DictContents	Contents of the dictionary values to be updated in the format: <variable name>\5<variable value>\5<variable format>\5<confirm get>

Response

FIELD	CONTENTS
Id	RSP_UPD_DICTIONARY
Type	response

8.13 Obtain the variables in procedure scope

Obtain the variable names and values (both global and local variables) currently in the procedure execution scope.

Request

FIELD	CONTENTS
Id	REQ_VARIABLE_NAMES
Type	request
VarGet	AVAILABLE_GLOBALS, AVAILABLE_LOCALS, AVAILABLE_ALL, REGISTERED_LOCALS, REGISTERED_GLOBALS, REGISTERED_ALL, NONE
CurrentChunk	Used only if the variable information is being divided into data chunks.

Response

FIELD	CONTENTS
Id	RSP_VARIABLE_NAMES
Type	response
CurrentChunk	Used only if the variable information is being divided into data chunks. Indicates the chunk being provided.
TotalChunks	Used only if the variable information is being divided into data chunks. Provides the total amount of chunks.
VarName	List of variable names separated by '\4'.
VarType	List of variable types separated by '\4'
VarGlobal	List of true/false flags separated by '\4' indicating if the variables are global
VarRegistered	List of true/false flags separated by '\4' indicating if the variables are registered

8.14 Change variables in procedure scope

Modify the value of one variable in the procedure scope.

Request

FIELD	CONTENTS
Id	REQ_CHANGE_VARIABLE
Type	request
VarName	Name of the variable
VarGlobal	True/False
VarValue	SPELL expression providing the value

Response

FIELD	CONTENTS
Id	RSP_CHANGE_VARIABLE
Type	response

8.15 Register interest for a variable in procedure scope

Register client interest on a variable in the procedure scope.

Request

FIELD	CONTENTS
Id	REQ_VARIABLE_WATCH
Type	request
VarName	Name of the variable
VarGlobal	True/False

Response

FIELD	CONTENTS
Id	RSP_VARIABLE_WATCH
Type	response

8.16 Unregister interest for a variable in procedure scope

Unregister the interest on a variable in the procedure scope

Request

FIELD	CONTENTS
Id	REQ_VARIABLE_NOWATCH
Type	request
VarName	Name of the variable
VarGlobal	True/False

Response

FIELD	CONTENTS
Id	RSP_VARIABLE_NOWATCH
Type	response

8.17 Unregister interest on all variables in procedure scope

Unregister interest on all variables in the procedure scope.

Request

FIELD	CONTENTS
Id	REQ_WATCH_NOTHING
Type	request

Response

FIELD	CONTENTS
Id	RSP_WATCH_NOTHING
Type	response

9 MESSAGES RECEIVED FROM PROCEDURES

The communication with procedures (i.e. executor processes) takes place through the Context process, which acts as a dispatcher. Therefore the same IPC interface used to interact with the Context is to be used to interact with procedures running within that context. The difference in the messages is that the message sender property indicates the procedure instance identifier.

9.1 Generic fields

All messages sent by procedures will have the following properties. These properties are not indicated in the following but they should be included in all one-way messages and requests.

FIELD	CONTENTS
Sender	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
Receiver	CLT
IpcKey	<executor IPC key>
ProcId	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0

All request responses sent to the procedure shall have at least the following fields:

FIELD	CONTENTS
Sender	CLT
Receiver	Procedure instance identifier (procId + instance number): e.g. Main/proc1#0
IpcKey	<client IPC key>
Type	response

Executors do not have the ability to process error messages.

9.2 Display messages

Text messages issued by the executors.

Message

FIELD	CONTENTS
Id	MSG_DISPLAY
Type	write (subclass of oneway)
Text	Text of the message
Level	Severity code: INFO, WARN, ERROR The values DEBUG, PROMPT are deprecated.
MsgType	DISPLAY, LOG, DIALOG, SCRIPT, UNKNOWN Kept for backward compatibility. Only DISPLAY should be used.
Time	String with the time when the message was originated.
Sequence	Sequence number providing order of message generation from executor. Used internally.
ExecutorMode	Indicates if the message was issued during a procedure or a manual execution. PROCEDURE or MANUAL.
Scope	Scope of the message. PROC, SYS, CFG, STEP, PROMPT or OTHER.

9.3 Data notifications

Data notifications that provide feedback about the operations ongoing in the procedure. They may be synchronous or asynchronous. If they are synchronous (type "notify") the client application shall send an ACKNOWLEDGE message back to the executor before processing the data. If they are asynchronous (type "notify_async") there is no need to send the acknowledge.

Message

FIELD	CONTENTS
Id	MSG_NOTIFICATION
Type	notify_async OR notify
DataType	Type of notification: CURRENT_LINE, CALL, RETURN, ITEM, STATUS
Time	Time when the message was created
Sequence	Sequence number providing order of message generation from executor. Used internally.

ExecutorStatus	Present for notifications of type STATUS. See executor status codes in REQ_EXEC_INFO.
ItemType	Present for notifications of type ITEM. It shall have value VALUE. Other values are kept for backwards compatibility and may dissapear in a future.
ItemName	Present for notifications of type ITEM. List of data item names, separated by ',,'.
ItemValue	Present for notifications of type ITEM. List of data item values, separated by ',,'.
ItemStatus	Present for notifications of type ITEM. List of data item status, separated by ',,'. Valid status codes are SUCCESS, FAILED, IN_PROGRESS, SUPERSEDED, UNKNOWN, SKIPPED, CANCEL
ItemReason	Present for notifications of type ITEM. List of data item comments, separated by ',,'.
ItemTime	Present for notifications of type ITEM. List of data item update times, separated by ',,'.
CodeName	Name of the current procedure source code. Used in CURRENT_LINE, CALL and RETURN notifications.
Csp	Current call stack. Used in all notifications except STATUS. The format is <code1>:<line1>:<code2>:<line2>… Each code identifier may correspond to a source code file or a function name within the source code files.
StageId	Used in CURRENT_LINE, CALL, RETURN notifications. Identifier of the current stage (Step) of the procedure.
StageTl	Used in CURRENT_LINE, CALL, RETURN notifications. Title of the current stage (Step) of the procedure.

9.4 Prompt start notifications

Indicate that a procedure is in PROMPT status and waiting for a user response.

Message

FIELD	CONTENTS
Id	MSG_PROMPT_START
Type	oneway
Text	Prompt message
DataType	Type of prompt. 16 (NUM), 2048 (COMBO) or other value.
Scope	Scope of the message. PROC, SYS, CFG, STEP, PROMPT or OTHER

OptionValues	List of available answers if any, separated by "|".
ExpectedValues	List of expected answer values if any, separated by "|"

9.5 Prompt end notifications

Indicate that a procedure is no longer in PROMPT status as an answer has been provided.

Message

FIELD	CONTENTS
Id	MSG_PROMPT_END
Type	oneway

9.6 Execution configuration changed

Indicates that the executor configuration has been changed.

Message

FIELD	CONTENTS
Id	MSG_EXEC_CONFIG
Type	oneway
<etc>	See request REQ_SET_CONFIG.

9.7 Variable changes

Sent by the executor when one or more **registered** variables change their value.

Message

FIELD	CONTENTS
Id	MSG_VARIABLE_CHANGE
Type	oneway
VarName	Variable name list, separated by '\4'

VarGlobal	True/False flags, separated by '\4'
VarValue	Value of the variables, separated by '\4'
VarType	Type of the variables, separated by '\4'