

An Introduction to SYMBION™ The Language

The `Symbion`™ language has been developed to provide a highly efficient, common interface for communicating with, networking, and controlling a wide range of analytical and other instruments. Its syntax is simple to use yet exhaustive, covering all of the significant capabilities required for instrument and sample system control, data analysis, I/O, database storage, and networking.

BENEFITS:

- One syntax for the control of all instruments
- One database for all measurement data (both spectral and time-dependent)
- One database for all instructions and administrative information.
- Easy extensibility
- Easy portability of configurations and data
- Ease of instrument and data validation
- A direct path to regulatory compliance

1. Introduction

The Symbion language is specifically tailored to provide a standard set of commands for the control of all portions of the analytical system while providing external interfaces to the next tier of data and control systems. The general arrangement is shown in Figure 1. Each command, C1 through C5, indicated in the figure is an object with predefined functionality that allows for interfacing with local and remote I/O, external chemometric software packages, analytical instruments, and internal and remote databases. The input and output arrangements allow for transparent interconnection of disparate sub-systems. This approach provides a powerful yet straightforward means for designing complex operating configurations and for linking existing programs into coherent units to meet evolving needs. In addition to the many instrumentation-specific commands developed for Symbion, the user can take advantage of extensive sets of readily available third-party commands.

The Symbion language underlies `Symbion-DX`™, a suite of programs for process development and on-line process control. (See Brochure PS:SDX-01.) This suite features a set of four main operating windows (programs) complete with pop-up windows and pull-down menus to facilitate most aspects of instrument operation and database management. The scripting capability is accessed through the Setup window that serves as the configuration builder.

This window features separate areas for initialization and run-time scripting, as well as a set of tools for use in composing, testing, and debugging script. It also

provides an interface to the Symbion console, providing direct interaction with the Symbion interpreter.

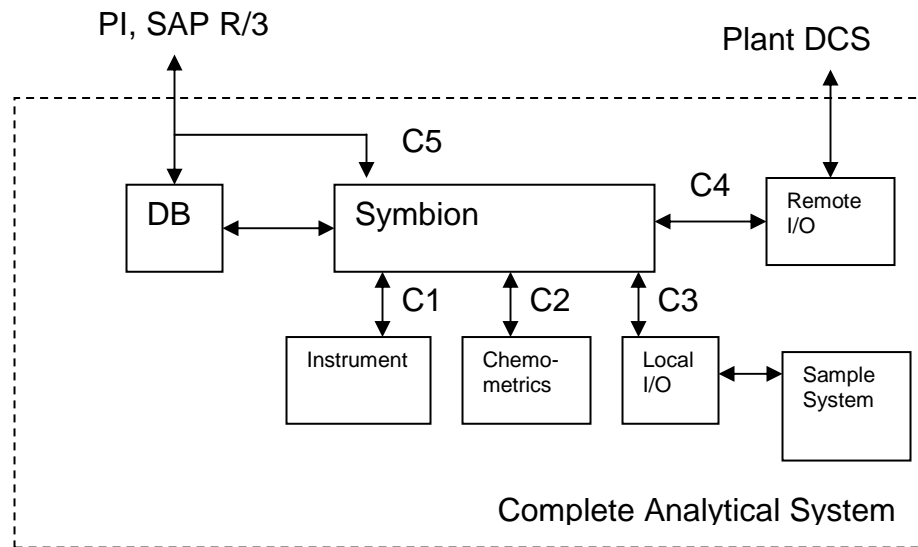


Figure 1 – Typical Data and Control Flow for Symbion™ Software

2. General Characteristics

The Symbion language is characterized by a simple yet powerful command syntax that translates the typical control objects into a standard set of hierarchically organized commands, allowing transparent control and fluid transport of data from one object to the next. The specific code related to the function of a given object is contained within that object, allowing the high level script to remain general and modular. The result is a high level of efficiency, economy, and transparency.

Symbion commands form a superset of the Tcl command set. As a result, Symbion commands can be augmented by the entire set of Tcl commands (www.scriptics.com). In addition, the Tcl extension architecture makes it possible for the user to build Visual Basic applications, C-code, or batch files and embed them seamlessly into Symbion. Furthermore, Symbion’s specific extension architecture provides a convenient and standard method for running user defined

instruments and I/O devices from Symbion commands. Substitution of a new device in a Symbion command will externally call a user’s program that contains the specific instructions for that device without affecting the other committed tasks inherent in the Symbion command.

Among the important characteristics of the Symbion language are a set of commands that standardize the functions and the input and output relationships between analytical instruments, analysis applications and industry standard I/O systems. Symbion converts these instruments, analysis applications and I/O systems into “black-box” objects that have pre-defined function, input, and output arrangements. The result is rapid application development.

Symbion’s embedding capability allows a wide range of existing code to be incorporated into Symbion commands. As a result, other scripting software packages can be run within Symbion, adding their capabilities to those of Symbion itself.

An additional benefit of the embedding capability is the security and tracking that it provides. When Symbion is operated with its Security Guard™ features enabled, any code embedded within the Symbion commands will share the security and traceability inherent in Symbion, even if the embedded code addresses a program that is not in itself secure. An example of this is given in Section 5.

3. Symbion Variable Format

In Symbion, instrument outputs and other data are transferred from one Symbion command to another as a structured format. Such “structures” comprise both numerical data and attribute information. For example, in the command

```
set x1 [ ControllInstrument Bruker Absorbance 16 None ],
```

the variable x1 references the name of the structure in which all of the numerical data and attribute information reside. Contained within the structure x1 are numerous objects, each associated with a particular associated datum or attribute. For instance, the data block xvar contained within x1 holds all of the x-axis data returned from the instrument. Similarly, the attribute block, ylabel, contained within x1 includes the y-axis label attribute returned from the instrument. A complete list of the data and attribute objects in the Symbion Programmers' Guide.

4. Symbion Syntax

The general syntactical format for a Symbion command is as follows:

```
set x1 [Command arg1 arg2 ...]
```

where **Command** = the Symbion command, *arg1* = first input argument, *arg2* = second input argument, and so forth, x1 is the variable name of output data. In the Symbion language, any expression wrapped

by [...] braces are evaluated by the interpreter and the result returned. The set command in Symbion works like an equals sign “=”. So, in the example command, the output of **Command** is placed in the variable x1. All Symbion commands return the name of a structure rather than actual data. Thus, the variable x1 is the name of the structure which contains the output from **Command**. Constructed in this way, any number of outputs can be placed into a single structure name and passed along to the input of another Symbion command. Many Symbion commands require an input in the form of a structure from a previous Symbion command. Thus, the output of one Symbion command can drive the inputs to subsequent Symbion commands.

5. Examples of Symbion Commands

Example 1:

In the following example, the output of a single Symbion command is connected to the inputs of two other Symbion commands. In this example data collected from a Bruker instrument is transferred to two different predictors: MATLAB and PLS/IQ, as shown in Figure 2.

The actual script that would make these connections is as follows:

```
set x1 [ ControllInstrument Bruker Absorbance 16 None ]
set x2 [ RunPrediction PLSIQ C:/Symbion/my.cal} $x1 0 ]
SendAnalysisData MATLAB spec $x1
SendAnalysisCommand MATLAB {
    y = spec(500);
}
set x3 [ GetAnalysisData MATLAB y ]
```

The first line instructs Symbion to command the Bruker instrument to take a 16-scan Absorbance. The data and other attribute from the sample collection using the Bruker spectrometer are placed into the structure referenced by the variable x1.

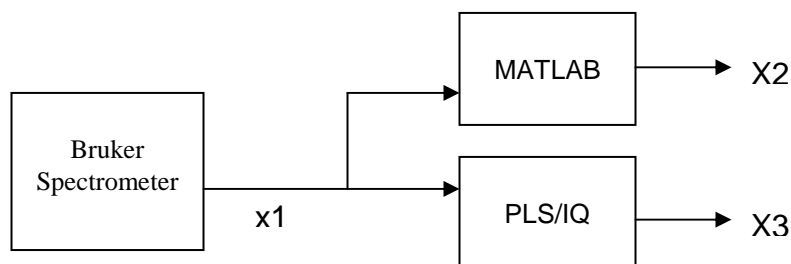


Figure 2 – Interconnection of Symbion Commands via Structures

To make use of the structure in other Symbion commands, the “\$” character must be placed in front of the variable x1, as in the second line. The second input argument of the RunPrediction command is the unknown spectral data to regress upon. Since the name of the structure is contained in the variable x1, a “\$” is necessary to reference the actual name of the structure. In the third line, the data from the variable x1 is again used to transfer Symbion data from the Symbion variable x1 to the MATLAB variable spec. Again, the third input argument of the SendAnalysisData command requires the name of the structure containing the spectral data, which is \$x1 and not x1 itself.

Once the data has been collected by the Bruker spectrometer and the resulting data is contained in the structure x1. The data can be passed to as many other Symbion commands as desired. The variable x1 forms an internal “Bus” for data (See Security Guard Architecture Reference). In the case of the PLS/IQ program, the Symbion driver for the PLS/IQ program hands the unknown spectral data from x1 and the calibration in C:/Symbion/my.cal to the PLS/IQ run-time engine for processing. After the processing is finished, the resulting data is returned into the structure x2 for storage, plotting, etc. In a similar fashion, the data from the x1 structure is sent to the MATLAB application; however, in this case, the MATLAB y-data in the variable x1 is mapped to a variable spec in MATLAB. Since MATLAB runs its own interpreter and uses its own scripting language, operations

performed on the newly created MATLAB variable y are run using the SendAnalysisCommand. The script inside the braces is passed to the MATLAB interpreter for evaluation. Once MATLAB completes, the following line of Symbion code is executed to map the result contained in the MATLAB variable y to the Symbion variable x3.

Example 2:

The following is an example of Symbion code that controls two instruments and sends their respective data to MATLAB for processing. The embedded MATLAB command contained within the “SendAnalysis” command will have the same traceability and security as the Symbion code.

```

set x1 [ ControllInstrument Bruker Absorbance 8 None ]
set x2 [ ControllInstrument Kaiser Sample 8 None ]

SendAnalysisData MATLAB bruker_spectrum $x1
SendAnalysisData MATLAB kaiser_spectrum $x2

SendAnalysisCommand MATLAB {

% Trivial Chemometrics for example purposes

peak1 = bruker_spectrum(500);
peak2 = kaiser_spectrum(250);
}

set x3 [ GetAnalysisData MATLAB peak1 ]
set x4 [ GetAnalysisData MATLAB peak2 ]

PlotRTTag structure $x3 Peak1 Concentration MATLAB
Percent black
PlotRTTag structure $x4 Peak2 Concentration MATLAB
Percent black
  
```

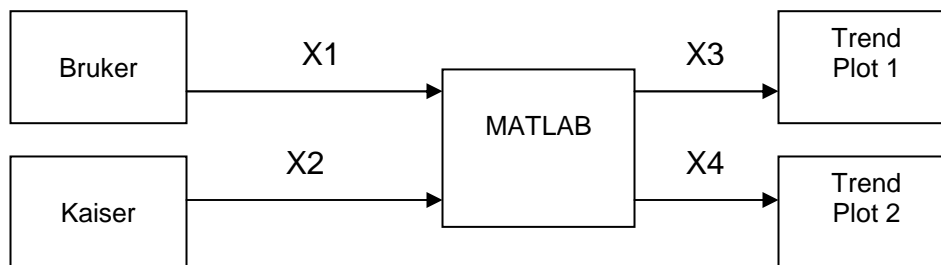


Figure 3 – Control of two Spectrometers with Data Processed in MATLAB

The final two blocks of code retrieve the processed data for display in the appropriate window.

6. The Programming Environment in

*Symbion-DX*TM

The *Symbion-DX*TM Process Analysis Suite is controlled from the Setup Window (See User Manual). The Setup Window provides overall system control and debugging of Symbion command scripts.

6.1 Initialization and Run-Time Script Windows

The main control scripts are located in the initialization and run-time panes as shown in Figure 4. The commands contained in the initialization script are run sequentially from top to bottom when the initialization button or pull-down is selected. This script is run at boot-up if auto-boot has been activated. The commands contained in the run-time script are run from top to bottom and then continuously looped over when the mode is switched to ONLINE by the mode selector buttons or pull-downs. By selecting the OFFLINE mode from either the OFFLINE buttons or pull-down, an OFFLINE request is submitted to Symbion. The current run-time script will run until completion and then switch to OFFLINE mode.

There are four additional script-related tools located in Figure 4 that assist in the programming and debugging of the Symbion

application. These are the composer, setup graph, setup messages and the console interface.

6.2. Composer Tool

The composer tool provides a scripting region separate from the initialization and run-time scripts for script development and debugging. The user can write control scripts into this window without affecting the main control scripts. A typical set of Symbion commands would first be developed in this scripting region and then run by pulling-down on the run menu option and then selecting *Execute Scripting in Symbion* in order to verify the proper operation of the code, the syntax, etc..

6.3. Setup Graph

The setup graph pop-up is available to plot data arising from the command scripts. Data plotted in this window will not conflict with any plot in the operational, manual or historical operations windows. This plotting region is mainly used as a debugging tool to view data that would normally be directed to the operational or run-time window. (See PlotXY, PlotRTSpec, and PlotRTTag commands in the command reference).

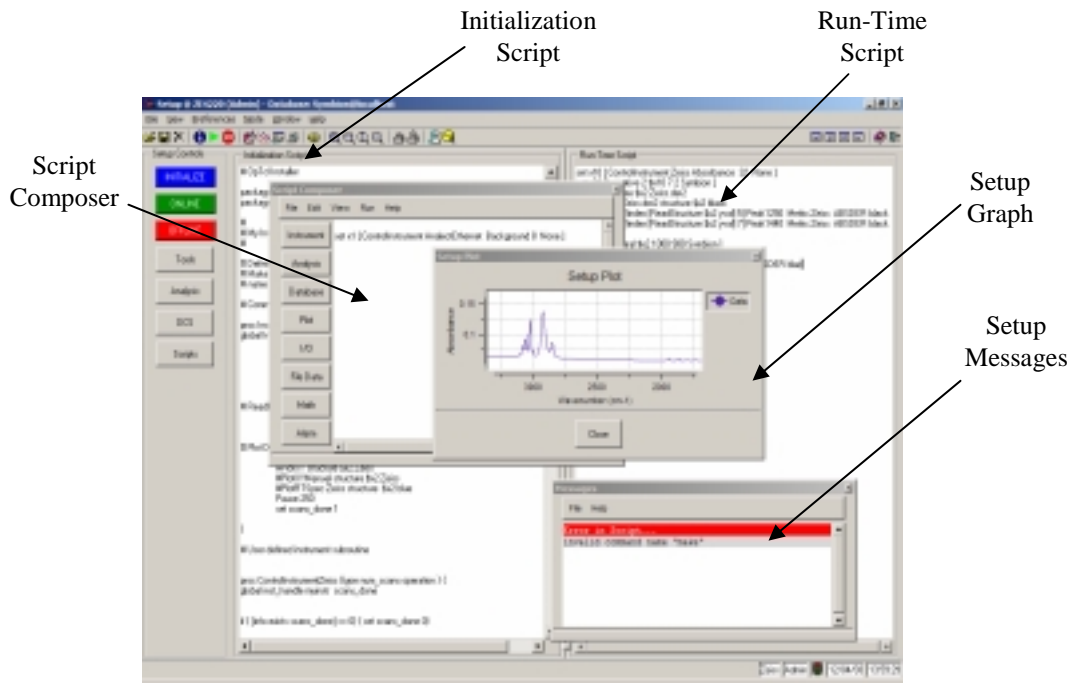


Figure 4 – S y m b i o n - D X TM Setup Window

6.4. Setup Messages

The setup message window displays errors and messages from the Symbion commands. General syntactical errors arising from the command scripts are also displayed for debugging purposes.

6.5. System Console

In addition to the pop-up windows from Figure 4, there is a Symbion console, which provides direct interaction with the Symbion interpreter.

7. Conclusion

The Symbion language provides a powerful yet economical set of tools for designing and operating diverse measurement configurations involving analytical instruments and other measurement and control devices. This document has provided a brief introduction to some of its capabilities. For a more detailed description, please refer to The Symbion Programmers' Guide, SD-502. For examples of Symbion implementation, see the S y m b i o n - D X TM brochure, PS:SDX-01.