EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN – AB DEPARTMENT

AB Note 2007-024 (RF)

Accessing accelerator parameters with Mathematica via the Passerelle

H. Damerau

Abstract

To access the control system of the accelerators in the PS Complex at CERN from an office PC running *Mathematica* in a Windows environment, the package `PasserelleUtils` has been implemented. This package is based on the functions provided by the *Passerelle*, a bridge between office PCs and the accelerator control system at CERN running Linux. It allows read and write access by executing synchronous function calls to accelerator parameters directly from *Mathematica*. The seamless integration of data from the accelerators into the powerful mathematical programming and graphics environment of *Mathematica* facilitates their analysis and post-processing. Parameter settings can be derived and sent to the accelerator equipment directly in *Mathematica*. Dedicated functions allow for easy read access to vector tables and sampler data as used to control and operate equipment in the PS Complex. Furthermore, it removes limitations inherent to the conventional way of using EXCEL to exploit the functions of the *Passerelle*. This note can be referred to as a manual of the `PasserelleUtils` package. After a brief introduction to its functionality, a detailed description of all functions provided is given. Several simple examples illustrate the usefulness.

Geneva, Switzerland

29 May 2006

1 Introduction

For the convenient analysis and post-processing of accelerator parameters they should be easily imported into an environment with powerful mathematical data handling and evaluation capabilities. One of such environments is the program *Mathematica* [1] which is widely used at CERN. Though not optimized for mass data processing, it offers a simple and intuitive way to analyze and display data with reasonably short code development time. Furthermore, high-level mathematical functionality, like numerical fit and integration routines, are available as well as the possibility of writing structured programs.

This note describes the basic functionalities of a versatile *Mathematica* extension package called `ControlSystemAccess‘PasserelleUtils‘` to access the accelerator control system of CERN by executing synchronous function calls directly from a *Mathematica* notebook running on a Windows PC. The package profits from the convenient data access provided by the *Passerelle* [2, 3, 4], the bridge between Windows computers in the office network and the technical network at CERN. Even though the *Passerelle* is not recommended for new applications anymore, it presently still represents the most simple access to equipment data from the office network.

If the control system access via the `PasserelleUtils` turns out to be useful, one should consider to implement a similar package based on calls of functions in JAVA to remain up to date with the future evolution of the accelerator control system at CERN.

2 Passerelle and PasserelleUtils

The *Passerelle* is a program in the form of the Windows DLL (Dynamic Link Library) `RPCAPI32.DLL` containing a collection of functions to read and write equipment properties of the accelerator control system at CERN. Its internal functionality is rather complex and out of the scope of this document [2]. The basic function to perform synchronous read or write operations (in C++ notation) is called `SyncRPC`. In the original form this function takes eight arguments to specify name, property and PLS line of the equipment concerned as well as some parameters to organize the data exchange between the DLL and the application. This is the main function used for equipment access with the `PasserelleUtils` package in Mathematica. Additionally, the DLL library of the *Passerelle* contains functions to establish hot-links between an application and a specific equipment in the control system. The hot-link functionality of the *Passerelle* is however currently not supported by the *Mathematica* package.

The *Mathematica* package `PasserelleUtils` basically allows the synchronous function calls implemented in the *Passerelle* DLL to be accessed with the *.NET/Link* package [5] in a convenient way. It transforms data from the DLL, i.e. from the control system, to a standard list structure. One could thus consider the `PasserelleUtils` package as a container in *Mathematica* around the corresponding functions of the *Passerelle* program.

The main functions of the `PasserelleUtils` package to read or write equipment data are called `SyncRPCRead` and `SyncRPCWrite`. They have less arguments than the original functions of the DLL library as some parameters are calculated automatically: name, property, PLS line and, in case of the write function, the data to be transferred to the equipment. The data handling, including variable types, type conversion and data length is either handled automatically by the *Mathematica* package or can be controlled manually by several options passed to the `SyncRPCRead` and `SyncRPCWrite` functions.

The data of a read access is passed in the form of a string or in the form of floating point variables in IEEE754 (double precision) format from the *Passerelle* routines to the *Mathematica* package. The same formats, strings for single values and doubles for lists, are also used for all write accesses to the control system. Lists of numbers are thus internally converted to arrays of double precision floating point numbers and passed to the *Passerelle*. This special format has been chosen as the transfer of a list of numbers in the string format is presently not supported by the *Passerelle*. With default options, the format conversions within the `PasserelleUtils` package are normally invisible to the user and the proper format is selected automatically.

Every time the `SyncRPCRead` function is called in *Mathematica*, the *Passerelle* DLL is connected via the *.NET/Link* features of *Mathematica* and the DLL library function `SyncRPC` is executed. This function accesses the equipment parameters. The resulting data is finally converted to a *Mathematica* digestible format and sent back to the *Mathematica* kernel. The write access is executed in the reverse order. The data to be transferred are prepared in *Mathematica*, passed via *.NET/Link* to the *Passerelle* DLL, and they are finally written to the equipment.

It is important to point out that the user of the `PasserelleUtils` package is not bothered with this internal functionality and may not even realize that *.NET/Link* connections are executed, the functions for equipment access of the `PasserelleUtils` package are integrated almost seamlessly into the *Mathematica* environment.

Furthermore, the access of control system parameters with the `PasserelleUtils` package and *Mathematica* removes certain restrictions inherent to the conventional way of using the *Passerelle* together with Microsoft EXCEL. Writing strings with more than 256 characters, e.g. for the description qualifier of linked timing trees, is presently not possible. However, as this is not due to a limitation of the *Passerelle* itself, much longer strings can be read and written by the `PasserelleUtils` package in *Mathematica*.

3 Using the PasserelleUtils package

3.1 Installation

Prior to the use of the `PasserelleUtils` package, an installation of the *Passerelle* library according to the description in [3] is required. Once the *Passerelle* is successfully installed, it is sufficient to add the directory \\afs\cern.ch\user\h\hdamerau\public\Mathematica\OwnPackages\ to the `$Path` variable of *Mathematica*. Then, the PasserelleUtils package is directly loaded by just typing `<<ControlSystemAccess‘PasserelleUtils‘` in a new *Mathematica* notebook.

3.2 Functions of the PasserelleUtils

The functions provided by `PasserelleUtils` package are explained in detail in what follows. This section may therefore serve as a reference manual. The default values of the options are given in the format `option → default value`.

3.2.1 Basic read and write access functions

- `SyncRPCRead[elementName,property,plsLine,options]`
  performs a read access to the equipment specified by the three parameters element name, property and plsLine. If the access is executed successfully, the result of the function contains the data that was asked for. If the equipment cannot be accessed, an error message is generated and the return value of the function is an empty list.

  A number of options can be given to the function call:

  - `ArraySize → 2501`
    represents the number maximum number of data read by the synchronous read access. This number should be larger than the maximum array length expected to be transferred from the equipment.
  - `SynchronousFunctionCallErrorMessage → True`
    specifies whether the detailed error message should be read from the *Passerelle* and shown as a message. If set to `False`, a general error message is generated in case of a problem.

  - `ReturnSingleValueList → False`
    specifies whether a single value result should be returned as a *Mathematica* list with just one single element {return value}. If set to `False`, a list is returned if the equipment property contains more than one value.
  - `ReadFromArchive → False`
    specifies archive data that should be accessed instead of the real control system when this option is set different from `False`. The option may contain the file name of an archive file to be accessed, or it may be directly set to a table of an imported archive file. The latter should be used if a larger number of equipment data are accessed as the archive file is newly imported for each access in the former case.
  - `ToExpressionConversion → True`
    specifies whether the the string received from the *Passerelle* should be converted to an expression or list digestible to *Mathematica*. If set to `False` the original string as received from the *Passerelle* DLL function call is returned. This option can be used for diagnostic purposes.
  - `MaximumReadTry → 10`
    specifies the number of successive read tries if an equipment is temporarily unavailable.
  - `PasserelleTransferDataType → "CF_TEXT"`
    specifies the data type used to transfer data via the Passerelle. Possible data type settings are `"CF_TEXT"` and `"CF_DOUBLE"`. The transfer of larger amounts of data is mostly faster and more stable using `PasserelleTransferDataType → "CF_DOUBLE"`.

- `SyncRPCWrite[elementName,property,plsLine,data,options]` or
  `SyncRPCWrite[elementName,property,plsLine,data`$_1$`,data`$_2$`,...,options]`
  performs a write access to the equipment specified by the three parameters element name, property and plsLine. The data can be a single value, a sequence or a list of numbers. If the access is executed successfully, the result of the function contains the data in the form of a byte list as sent to the *Passerelle*. Depending on the transfer format, this byte list contains character codes of a string or a list of doubles stringed together, each represented by eight bytes. If the equipment cannot be accessed or the data does not match the requirements of the equipment, an error message is generated and the return value of the function is an empty list.

  A number of options can be given to the function call:

  - `SynchronousFunctionCallErrorMessage → True`
    specifies whether the detailed error message should be read from the *Passerelle* and shown as a message. If set to `False`, a general error message is generated in case of a problem.
  - `ToStringConversion → True`
    specifies whether the the data should be converted to an appropriate string expression before transferring them via the *Passerelle*. If set to `False` the data must contain an already prepared string to be transferred directly. This option can be used for diagnostic purposes.
  - `MaximumWriteTry → 10`
    specifies the number of successive write tries if an equipment is temporarily unavailable.

3.2.2 Read access functions

- `ReadSamplerData[elementName,plsLine,options]`
  reads several equipment properties relevant to get the physical information from a sampler device. The information is returned in the form of a header consisting of a list of pairs {property name,value}. The last element of the list represents the recorded data trace in the form of a point list {time in ms,value}.

  - `SamplerDataHeader → True`
    specifies whether the header information is returned. If set to `False` only the data trace is returned.
  - `SamplerDataUnit → "CalculatedPhysical"`
    specifies the units of the data trace returned. This option is only active for samplers of the `SAMPC` type. If set to `"CalculatedPhysical"` the data is read in hardware function units and scaled with the scaling parameters. The trace is returned in physical units. If set to `"HardwareFunction"` the data is returned in units of the hardware function. If set to `"Physical"` the data is directly read and returned in physical units. This might cause problems when reading more than some 730 array entries.
  - `SamplerType → "SAMPC"`
    selects the sampler type from which the data is read. Possible settings are `"SAMPC"`, `"SAMPC-A"`, `"SAMPC-B"`, `"SAMPC-C"`, `"SAMPC-D"`, `"SAMPC-E"` or `"SAMPAQ"`, `"SAMPAQ-A"`.
  - `SamplerArray → "ARRA1"`
    specifies the data array read from a type `"SAMPAQ"`, `"SAMPAQ-A"` sampler device. This option is inactive for all type `"SAMPC"` samplers. Possible settings are `"ARRA1"`, `"ARRA2"` and `"ARRA3"`.
  - `SamplerDataScaling → False`
    specifies whether the sampler data should be converted. Possible scaling settings are `"LogarithmicToLinear"` and `False`. This option is only active for all type `"SAMPC"` samplers.

- `ReadVectorTable[elementName,plsLine,options]`
  reads several equipment properties relevant to get the physical information from a GFAS vector table. The information is returned in the form of a header consisting of a list of pairs {property name,value}. The last element of the list represents the vector table in the form of a point list {time in ms,value}.
  - `VectorTableHeader → True`
    specifies whether the header information is returned. If set to False only the vector table is returned.
  - `VectorTableUnit → "Physical"`
    specifies the units of the vector table returned. Possible settings are `"Physical"` and `"HardwareFunction"`.
  - `VectorTableScaling → False`
    specifies whether the vector table should be converted. Possible scaling settings are `"LogarithmicToLinear"` and `False`.
- `ReadEventCTime[elementName,plsLine]`
  returns the time in ms from C0, the so-called C time of an event.

3.3 Auxiliary functions

- `GetRPCLastError[]`
  returns a description string of the last error.
- `VectorTablePlot[vectorTableData,options]`
  generates a plot of a vector table list as generated by a call of the `ReadVectorTable` function. `VectorTablePlot` takes most of the options of `ListPlot`.
  - `VectorTablePlotMode → "SequenceInternalStop"` specifies the representation of a vector table with internal stops. Possible plot mode settings are `"Sequence"`, `"SequenceInternalStop"`, `"Overlay"` and `"OverlayInternalStop"`.

- `SamplerDataPlot[samplerData,options]`
  generates a plot of a sampler data list as generated by a call of the `ReadSamplerData` function. `SamplerDataPlot` takes most of the options of `ListPlot`.
- `UnixEpocheToDate[unixEpocheTimeStamp]`
  converts an absolute time in seconds since the beginning of January 1, 1970 to a date in the form a of a list {year,month,day, hour,minute,second}.
- `UnixEpocheToDateString[unixEpocheTimeStamp]`
  converts an absolute time in seconds since the beginning of January 1, 1970 to a date in the form of a string.
- `GetFromVariableList[variableList,variableName]`
  gets the variable named variableName from a list of variable name,value. This function can be used to extract information from a typical header list.

4 Further packages to facilitate the equipment access

Besides the `PasserelleUtils` package itself, a number of smaller packages using the control system access functions have been written. These dedicated packages (presently `CPSRFParameterUtils.m`, `PSBRFParameterUtils.m` and `ArchiveUtils.m`) can also be found in the \\afs\cern.ch\user\h\hdamerau\public\Mathematica\OwnPackages\ControlSystem-Access\ folder. They provide only a small number of functions and, as a short help text is available for each function, their use should be self-explanatory. The package `CPSRFParameterUtils.m` serves to extract the matrices of the PS 10 MHz and PS 200 MHz RF systems. The package `PSBRFParameterUtils.m` allows a comfortable access to parameters of the PS booster as it provides functions to read data from all four rings automatically and compare them in a comfortable way. The `ArchiveUtils.m` package presents a first attempt of using the `PasserelleUtils` package to store a certain number of parameters in an archive file which can be later on accessed by the `SyncRPCRead` function as a virtual control system on disk.

5 Examples

In what follows, a few short examples of the usage of the various functions of the `PasserelleUtils` are given. All of them are directly copied from a Mathematica notebook and should run on every office PC in the CERN network. Please note that write access is only granted to registered users of the *Passerelle*.

5.1 Basic read and write access

The *Mathematica* output of a simple read and write access to the number of frame triggers for the oscilloscope used for the Tomoscope in the first ring of the PS booster is shown in Fig. 1. After the `PasserelleUtils` package is loaded, the number of triggers is first read from the equipment, then set to 500 trigger signals and then set back to its original value.

The property `"CTLSTAMP"` of the equipment returns a time stamp in UNIX epoche format (for some equipment it may also be ten times the number of seconds defined by the UNIX epoche) when the last write access has been performed. As can be easily seen, time and date of the last write access indicated by the equipment are indeed consistent with the time when the *Mathematica* notebook was executed.

```
In[1]:=  << ControlSystemAccess`PasserelleUtils`;

In[2]:=  SyncRPCRead["BAX1.TOMOBURST", "NMEAS", "AD"]

Out[2]=  100

In[3]:=  SyncRPCWrite["BAX1.TOMOBURST", "NMEAS", "AD", 500]

Out[3]=  500

In[4]:=  SyncRPCRead["BAX1.TOMOBURST", "NMEAS", "AD"]

Out[4]=  500

In[5]:=  SyncRPCWrite["BAX1.TOMOBURST", "NMEAS", "AD", 100]

Out[5]=  100

In[6]:=  SyncRPCRead["BAX1.TOMOBURST", "NMEAS", "AD"]

Out[6]=  100

In[7]:=  Date[]

Out[7]=  {2006, 5, 29, 18, 0, 48.2205877}

In[8]:=  IntegerPart[SyncRPCRead["BAX1.TOMOBURST", "CTLSTAMP", "AD"]]
         UnixEpocheToDateString[%]

Out[8]=  1148918448

Out[9]=  29/05/2006, 18:00:48
```

Fig. 1: Basic read/write example using the `PasserelleUtils`.

5.2 Read and write a vector table

Not only a single value can be passed as data for a write access but also an array of data can be transferred to equipment having a property with more that one element. The GFAS (Simple Analog Function Generator) [6] is such a device being widely used in the RF systems of the machines in the PS Complex. An excerpt from a *Mathematica* notebook to illustrate a typical read and write access of a vector table is shown in Fig. 2. The first part of the package shows how an arbitrary data array representing eight vectors is written to a voltage program of the 200 MHz RF system used in the PS. The second part demonstrates how to copy the vector table information `PA.GSVREB1` into `PA.GSVREB2`. Please note that the zero at the end of the list is required for the GFAS to understand the data array as a vector table.

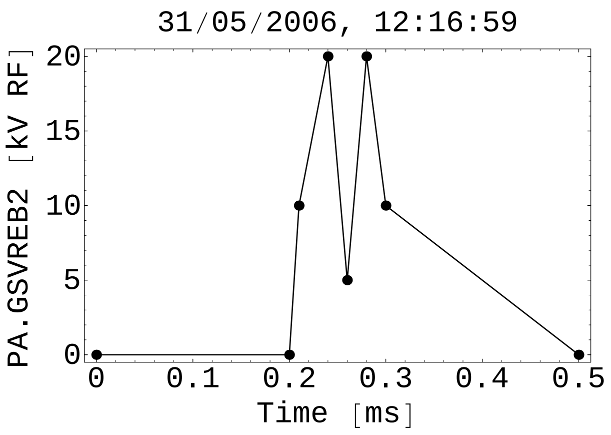
5.3 Read vector tables and sampler data

The third example illustrates the use of the read access functions to retrieve a vector table and the corresponding sampler data. The excerpt from the *Mathematica* notebook is shown in Fig. 3. After loading the `PasserelleUtils` package, the vector table of the relative phase program between the C02 and the C04 RF system of the first booster ring is read. The resulting list structure can be directly taken as an argument to the `VectorTablePlot` function to plot the vector table. Thereafter, the same phase parameter is read from the sampler device using `ReadSamplerDate`. The returned list structure contains a header similar to the information retrieved from the vector table read and a trace consisting of approximately 1000 data points. Again, the list structure is given as an argument to the dedicated plot function which generates the second plot shown in Fig. 3.

```
In[1]:=  << ControlSystemAccess`PasserelleUtils`

In[2]:=  SyncRPCWrite["PA.GSVREB2", "CCV", "SFTPRO",
           {8, 0, 0, 0.2, 0, 0.21, 10, 0.24, 20, 0.26, 5, 0.28, 20, 0.3, 10, 0.5, 0, 0}];

In[3]:=  VectorTablePlot[ReadVectorTable["PA.GSVREB2", "SFTPRO"]];
```



```
In[4]:=  SyncRPCWrite["PA.GSVREB2", "CCV", "SFTPRO",
           Append[SyncRPCRead["PA.GSVREB1", "CCV", "SFTPRO"], 0]];

In[5]:=  VectorTablePlot[ReadVectorTable["PA.GSVREB2", "SFTPRO"]];
```

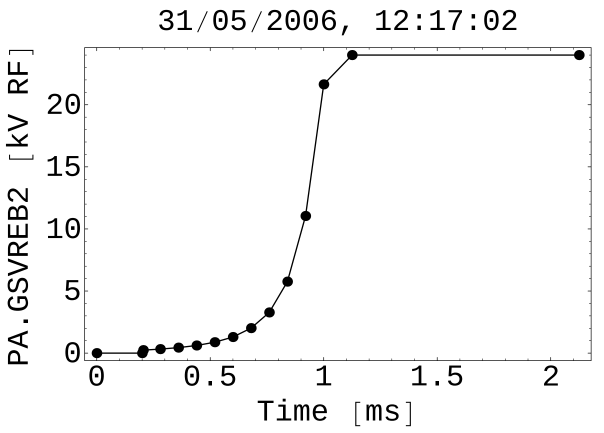


Fig. 2: Writing a data array to a GFAS equipment using the `PasserelleUtils`. The upper plot shows an arbitrary voltage function written to the GFAS `PA.GSVREB2` for the second re-bunching in the PS (normally not in use). The lower plot illustrates the operational voltage program for re-bunching copied with the `PasserelleUtils` from `PA.GSVREB1` to `PA.GSVREB2`.

5.4 Read and decode the cavity matrices

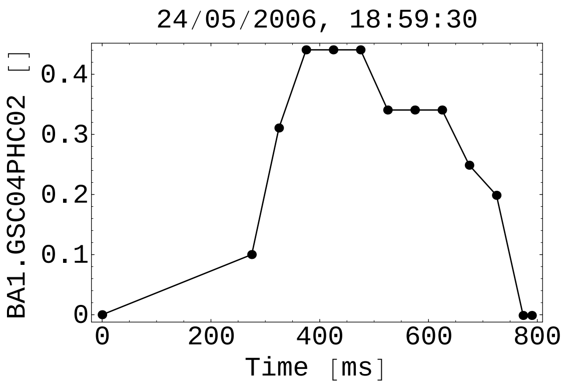
The voltages and timings of the 10 MHz and 200 MHz RF systems in the PS are controlled by cavity matrices routing voltage programs and timing pulses to a specific set of cavities defined in the matrix. As the underlying hardware is different for the 10 MHz and 200 MHz system, the cavity matrices are stored by the control system in different formats. A dedicated function of the extension package `CPSRFParameterUtils` allows to read and decode them into a unified list structure as shown in Fig. 4. Options can be given to specify whether table headings should be included, as in the example, or if they should be suppressed.

```
In[1]:=  << ControlSystemAccess`PasserelleUtils`;

In[2]:=  ReadVectorTable["BA1.GSC04PHC02", "ISOHRS"]

Out[2]=  {{CCVTRM, 0}, {DATE, 1148489970}, {DELAY, 275}, {ENABLE, 1}, {FUNCT, 24},
          {HIGH, 31}, {MASTER, 0}, {MAXV, 10}, {MINV, -10}, {MODE, 0}, {POLAR, 0},
          {REFVAL, 0.440687}, {SCL1, 1}, {STATUS, 216}, {BUS, 2}, {UNITA, }, {UNITS, },
          {CTLSTAMP, 1.14899×10^9}, {EMNUM, 172}, {EQNAME, BA1.GSC04PHC02},
          {{0, 0}, {275, 0.100101}, {325, 0.310678}, {375, 0.440687}, {425, 0.440687},
           {475, 0.440687}, {525, 0.340587}, {575, 0.340587}, {625, 0.340587},
           {675, 0.248726}, {725, 0.198675}, {774, -0.00122074}, {790, -0.00122074}}}

In[3]:=  VectorTablePlot[%];
```



```
In[4]:=  SamplerDataPlot[ReadSamplerData["BA1.SAC02PHC04", "ISOHRS"], PlotRange → {-2.5, 10}];
```

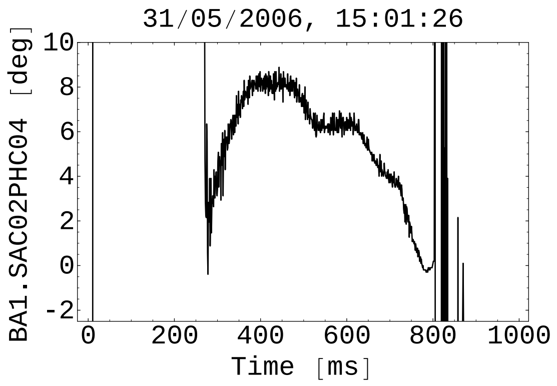


Fig. 3: Read access function example using the `PasserelleUtils`. The vector table `BA1.GSC04PHC02` of the programmed phase between the two RF system at harmonic number two and four is illustrated in the upper plot. The lower plot shows the same parameter as measured with a sampling rate of 1 kHz by the sampler `BA1.SAC02PHC04`.

5.5 Reading equipment data from all four PS booster rings

The last example shows a combined read access to retrieve sampler data from all four rings of the PS booster (Fig. 5). The functions used are not directly included in the `PasserelleUtils` package but can be found in the extension package `PSBRFParameterUtils.m`. It is worth noting that the `PasserelleUtils` is automatically loaded when one of the extension packages is accessed. The function `ReadSamplerDataAllRings` gets the sampled RF amplitude in logarithmic units. In these units one corresponds to 20 dB with respect to 1 kV, that is 10 kV; minus one thus corresponds to 100 V, etc. The result of the function call is directly passed to a special plot routine to visually compare the data with those of the other booster rings.

```
In[1]:= << ControlSystemAccess`CPSRFParameterUtils`;

In[2]:= CavityMatrix200MHz["TSTLHC"] // TableForm

Out[2]//TableForm=
       Cavity:             201      202      203      204      205      206
       LE Blow-up          1        1        1        0        0        0
       3.5 GeV Blow-up     0        0        0        0        0        0
       HE Blow-up          0        0        0        0        1        1
       MD Blow-up          0        0        0        0        0        0
       CT Rebunching       0        0        0        0        0        0
       SE Channelling      0        0        0        0        0        0

In[3]:= CavityMatrix10MHz["TSTLHC"] // TableForm

Out[3]//TableForm=
       Cavity:       11      36      46      51      56      66      76      81      86      91
       Modif. 1      0       0       0       0       0       0       0       0       0       0
       Modif. 2      1       0       0       0       0       0       0       0       0       0
       Modif. 3      0       0       0       1       0       0       1       1       0       1
       Modif. 4      0       0       0       0       1       1       0       0       0       0
       Modif. 5      0       0       0       0       0       0       0       0       1       0
       Modif. 6      0       1       1       0       0       0       0       0       0       0
```

Fig. 4: Automatic decoding of the cavity matrices of the 10 MHz and 200 MHz RF systems using a function of the `CPSRFParameterUtils` extension package.

6 Conclusions

The *PasserelleUtils* package described in this note provides an easy read and write access to control system equipment of the PS Complex at CERN. The access is performed via the infrastructure of the *Passerelle*, a bridge between the office PCs running Windows and the control system running Linux. The new functions of the package integrate seamlessly into the *Mathematica* environment, as the *Passerelle* DLL is connected to *Mathematica* via its *.NET/Link* functionality.

The author is grateful to François Chevrier, Eugenia Hatziangeli, Steven Hancock, Nicolas de Metz-Noblat and Javier Serrano for helpful discussions and information on the *Passerelle*, as well as on the equipment of the control system. Valuable comments by Andy Butterworth, Edmond Ciapala and Elena Shaposhnikova are also acknowledged.

References

[1] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, Redwood City, California, 1988

[2] I. Deloose, *Simultaneous access to the Controls of the PS & SL machines from the Windows 95 and NT Platforms via PS & SL passerelles, User's Guide.* PS/CO Note 98-33 (Tech.), CERN, Geneva, Switzerland, 1998

[3] F. Chevrier, *New RPCAPI32 DLL User Document.* Internal note, CERN, Geneva, Switzerland, 2005

[4] F. Chevrier, E. Hatziangeli, *Passerelle.* Web page of the Passerelle project,
`http://proj-passerelle.web.cern.ch/proj-passerelle/`

[5] T. Gayley, *.NET/Link User Guide, Version 1.2.* Software documentation, Wolfram Research, Champaign, Illinois, 2004

```
In[1]:=  << ControlSystemAccess`PSBRFParameterUtils`

In[3]:=  Show[
           GraphicsArray[Partition[CompareAllRingsSamplerDataPlot[ReadSamplerDataAllRings[
               "BAx.SAVRFDC02", "ISOHRS"], DisplayFunction → Identity], 2]], ImageSize → 600];
```

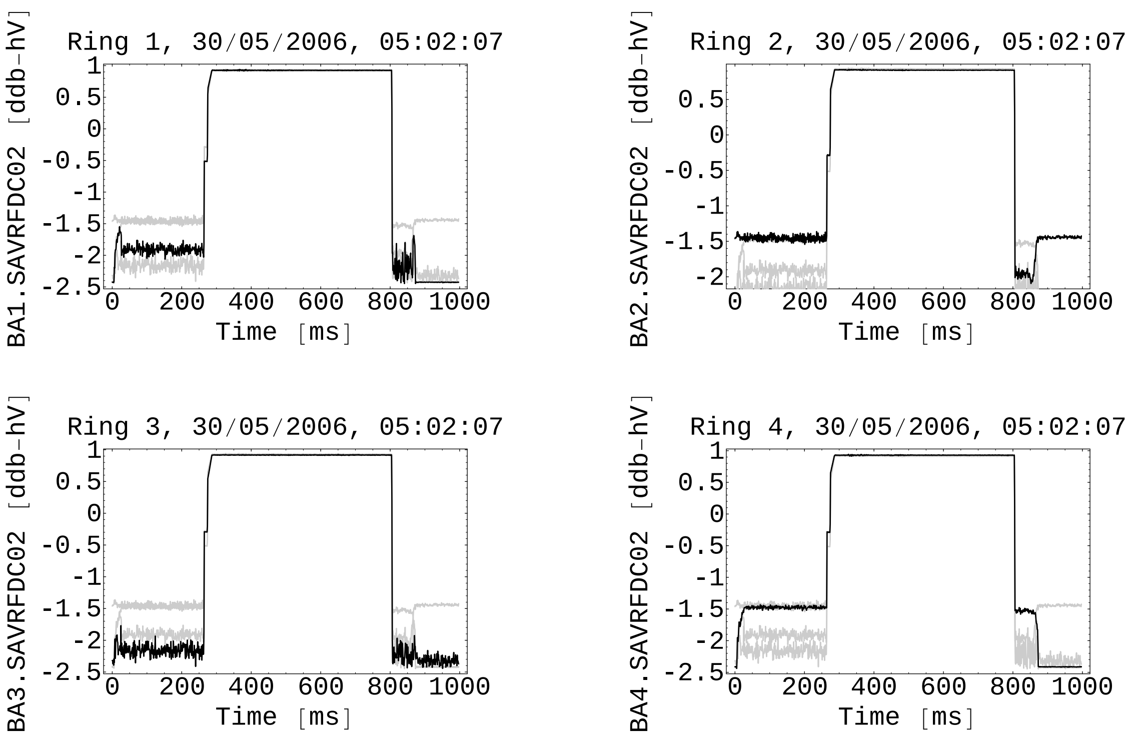


Fig. 5: Multiple read access on all four PS booster rings using the `PSBRFParameterUtils` package. In this example the RF voltage of the main RF system as measured by the samplers `BA1.SAVRFDC02` to `BA4.SAVRFDC02` is plotted in logarithmic units for comparison of the different rings.

[6] W. Heinze, *The Equipment Module for the GFAS*. PS/CO Note 93-108 (Tech.), CERN, Geneva, Switzerland, 2001