HIERACHIES IN EPICS AND VISUAL DCT

M. Sekoranja, I. List, R. Sabjan, S. Sah, I. Verstovsek, Cosylab, Ljubljana, Slovenia
J. Maclean, APS, Argonne National Laboratory, Illinois, USA

Abstract

Visual DCT [1] (Visual Database Configuration Tool) is becoming the most popular graphical database configuration tool for EPICS [2] databases. EPICS is a widely used control system based on a real-time database configured from text files. The configuration data comes from two types of files - one defines templates and the other instantiates channels from them. The current EPICS template substitution mechanism [3] is restricted in its capabilities and it only allows macros to be passed downwards into a template instance. This limitation makes an EPICS database totally flat, which can result in great difficulties when designing complex applications, not to mention maintaining them. The next release of EPICS will introduce hierarchy support into EPICS core. Instruments will be provided for templates to export fields that are of public interest and thus resembling mechanisms of object-oriented programming. However, this release may still be some time away, while developers need those features now. The present version of Visual DCT therefore has full graphical support for hierarchical features, enabling developers of today to use the technology of tomorrow. VDCT leads the field in advanced features, such as reverse engineering of already existing databases, and user friendliness. It is also the only graphical database configuration tool, which not only supports hierarchies in the EPICS databases now, but also incorporates an integral Database Flattening Tool to produce EPICS databases to be compatible with and executable on current core releases. New development will also introduce debug tools allowing real-time display of channel properties inside Visual DCT, making database debugging much easier.

EPICS AND HIERARCHIES

Abstraction of certain algorithms and grouping functions to make logical blocks have helped computer programming immensely. Only by using these approaches can a programmer of today cope with the complexity of modern programs, understanding those thousands of lines of source code.

EPICS core so far has no support of hierarchical programming. Developers worldwide use different approaches how to cope with complex EPICS databases. Up until now the best tool to use was CapFast [4], which is a commercial tool for making electronic drawings, adapted to support EPICS databases as well. The main disadvantage of CapFast is its price, which is considerably high, especially for mostly academically based EPICS community. Developed for a different purpose, CapFast is not optimized for EPICS databases as well.

HIERARCHIES AS A PART OF EPICS

EPICS community understands the problem and lack of hierarchical designs inside EPICS databases. The next major release of EPICS core will therefore include support for hierarchical designs. This calls for additional keywords in the EPICS database syntax.

So far, there were two types of file associated with EPICS databases. First, we had a template file in which the database was actually "programmed" and a substitution file which was able to create multiple instances of templates, filling macros with appropriate values.

The new proposal allows for a two-way communication among different templates. This includes passing down macros into the template instance (giving values for fields within expanded template) and values to be exported from the template instance to the higher level (usually the destination field name for a link in a record defined in the higher level file).

Macros are defined in the **expand** statement and pass information into a template; ports are a kind of macros defined in a **template** statement that pass information upwards out of a template instance to their calling database.

```
record(calc,"slide1:error") {
  field(INPA,"$(slmot1.position)")
          ...
}
expand("slideMotor.vdb", slmot1) {
  macro(name, "sm1")
  macro(address, "4")
  macro(demand, "slide1:demand.VAL")
}
record(ao,"slide1:speed") {
  field(OUTP, "$(slmot1.speed)")
  field(DTYP, "Soft Channel")
  ...
}
```

Figure 1: Example of usage of *expand* statement used in a higher level file.

Note that we are using the macro syntax **$(template_instance.port_name)** to bring port values from templates into the higher level diagram. Unfortunately we have to allow port macros to be used before the related expand statement appears in the parent file, so any database flattening tool will have to make two passes through the data and should also detect loops in port/macro definitions.

```
template("Description of the Slide
Motor template...") {
  port(speed, "$(name):speed.VAL",
"Record to set motor speed mm/sec")
  port(go, "$(name):startmoving",
"Forward link to this to cause
movement")
  port(position, "$(motor.position)",
"Current position of the slide")
  port(greet, "Hello, world!", "Just
being friendly...")
}
record(ai, "$(name):speed") {
          ...
}
```

Figure 2: Example of the *template* statement in a lower level file (slideMotor.vdb).

When performing macro substitutions within strings, if a macro name is undefined the macro name and its surrounding **$()** characters will be left unchanged in the flat .db file. This allows templates to be used when creating a database that still takes macro arguments on loading with **dbLoadRecords()**. For undefined port macros though an error should probably be reported instead (but remember that these can't be properly checked and substituted until all expand statements and their related templates have been read in.

TEMPLATES AND VISUAL DCT

Visual DCT is the fast evolving Database Configuration Tool of today [5, 6]. It was designed especially for EPICS to provide features needed by EPICS developers. It is written in Java and runs on a number of platforms.

Recent developments introduced the aforementioned hierarchical features into Visual DCT templates with full graphical support. Ports and macros can be defined with a mouse click and linked to appropriate process variables just like normal links.

User has to determine the port/macro type. This affects the representation of ports and macros in the database.

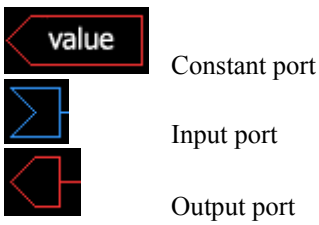

Figure 3: Representation of different port types in the lower level file. Macros are displayed similarly.

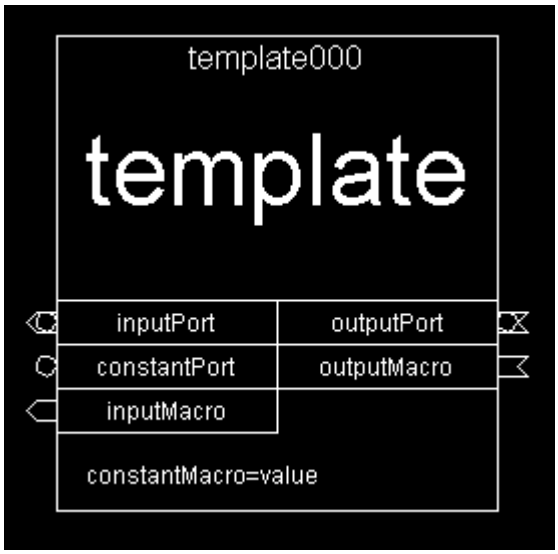

Figure 4: Example of expanded template in a higher level file.

Due to the fact that hierarchies are not yet a part of EPICS core, files created with Visual DCT and using hierarchical designs should have a special extension (**.vdb**), which is changed to a normal extension (**.db**) when flattened by a Database Flattening Tool.

DATABASE FLATTENING TOOL

In order to be able to use EPICS databases produced by Visual DCT on current EPICS core, a special Database Flattening Tool was produced and integrated into Visual DCT. The result of the process is a flat EPICS database which can then be loaded into an IOC.

The flattening process involves expanding all templates and replacing the macro and port macro variables with their strings. If a macro name is found that has no definition within its scope, it will be left exactly as it was found, which allows load-time macros to be used. The Database Flattening Tool also puts comments to the flat database file at the start and the end of each expanded template. This provides a way for any other tool to refer back to original template from the flat file:

```
# expand("/full/path/to/template.vdb",
instance_name)
  ...expanded contents of template.vdb
# end (instance_name)
```

Figure 5: Flattened database and reference to original template.

CAPFAST CONVERSION TOOL

A conversion tool which converts CapFast drawings to Visual DCT supported files, with practically no manual interventions, is also being produced. Several features are incorporated within, such as automatic conversion of drawings, all the functionality, and conversions of hierarchies.

Different shapes and sizes are used for records in CapFast which are not compatible with Visual DCT, which uses only rectangular shapes. So there is some compensation on this account and usually drawings have to be manually adjusted. But considering this and all the features we get by using Visual DCT it is just a small turn-off.

All the functionality is already automatically converted, except there are still some efforts put into producing the same results as CapFast does.

While converting hierarchies CapFast introduced some new problems, due to its prime intention to make electronic drawings. CapFast could not tell the difference between macros and ports, which is the main problem, and therefore the whole hierarchy has to be examined before a single Visual DCT file can be produced.

The final goal of this project is to generate Visual DCT files which in turn produce the same EPICS databases as CapFast does.

JCA DEBUG PLUGIN

A special Visual DCT plug-in was also produced which introduces new debugging possibility to EPICS databases developers. The developer can load a database into an IOC and observes the PV values from the Visual DCT. Only a substitution file has to be provided to cater the values of all macros.

The VAL field is displayed with larger fonts together with its timestamp. The developer can select which other fields are monitored and displayed through the inspector dialog.

The debug plug-in also supports EPICS alarms, colouring the values accordingly. Timeouts are also indicated with a clock sign displayed over the record instance symbol.

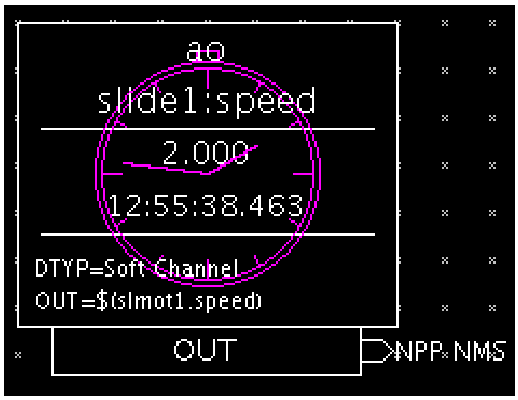

Figure 6: Timeout is indicated for the *slide1:speed* record. Last value received was 2.000 at 12:55:38.463.

Communication between Visual DCT and the IOC is achieved using Java Channel Access [7].

CONCLUSION

A large portion of EPICS community still uses text editors to configure their databases. This can leads to control system applications that are hard to maintain and expand. Though EPICS scales incredibly well as a control system, the development process can become a major bottleneck, requiring too much time and resources. This can be avoided with using (graphical) database configuration tools. Other concepts, such as hierarchical designs, furthermore divide templates into more readable and maintainable chunks.

By implementing hierarchies Visual DCT has bridged the last big gap towards the competition. Taking into account the fact that Visual DCT is already very popular among EPICS community and considering extreme extensibility and development rate of Visual DCT, we can justifiably claim that it truly represents the future of EPICS database configuration.

ACKNOWLEDGEMENTS

We would like to express out gratitude to all people involved on Visual DCT project, especially the funders of certain packages.

Our thanks go to APS, Diamond and SNS for funding the development of hierarchical features in Visual DCT. This is truly the most important improvement and a large step forward.

Once again we would like to thank SLS for supporting the development of JCA Debug Plug-in. The first version of Visual DCT was a result of ideas from SLS and their funding and we are very happy that they remain committed to the success of this product.

JLab has kindly encouraged the development of CapFast to Visual DCT Conversion Tool. We expect the final version in the months to come.

REFERENCES

[1]http://visualdct.cosylab.com
[2]http://www.aps.anl.gov/epics
[3]http://www.aps.anl.gov/epics/extensions/msi/index.php
[4]http://phase3.com/epics.html
[5]M. Sekoranja et al., “Visual DCT – Visual EPICS Database Configuration Tool”, ICALEPCS 2001
[6]R. Sabjan et al., “Visual DCT – EPICS Database Can Be Fun”, PCaPAC 2002
[7]http://www.aps.anl.gov/xfd/SoftDist/swBCDA/jca/jca.html