CATT DLL Directivity Interface (DDI) v1.0

DDI White Paper

Rev. 0, 981021 (first public version)

www.netg.se/~catt

Contents

1. Introduction

1.1 Abbreviations used

pred. : prediction software, e.g. *CATT-Acoustic*

manuf. : manufacturer of loudspeaker / provider of interface

1.2 The DDI goal/purpose

First, as a reminder, a WIN32 DLL (Dynamic Link Library, sometime called a plug-in) is basically like any other program running under *Windows 95*, *Windows NT* or higher but that does not have to be linked in and loaded together with an executable (EXE-file). A DLL can be loaded at will to perform specific tasks e.g. based on user selections. A common text-book example of how DLLs may be employed is if someone wants to create a terminal emulator. Depending on the users' choice of terminal, say VT220, only the VT220 DLL needs to be loaded instead of all available emulators having to be linked in creating a huge EXE-file. It also makes it easier to upgrade specific parts of a software if they are created as DLLs.

Further, for many tasks **including the DDI described here**, the DLL programmer does not have to learn anything about programming the WIN32 graphical user interface (GUI). Standard programming skills will suffice and only some language/compiler-dependent specific compiler and linker directives need to be set properly so that a WIN32 DLL is created instead of an EXE.

Purpose and goals of the *CATT* WIN32 *DLL Directivity Interface* (*DDI*) are:

- To provide a unified interface towards any kind of directivity be it measured, simulated or analytically described and at virtually any angular and frequency resolution. Rather than a complex file format the DLL interface is based on a query principle: The *pred.* first asks the DLL what kind of data, resolutions and options it supports and then decides, based on what it can handle itself, what it will use and then tells the DLL so.

  To create a file-format with the same flexibility would be very difficult and would take a very long time to agree on. Such a file format will also be very difficult to use while still not allowing for runtime simulation of e.g. arrays. A good example of a "too" complex file-format is the RIFF-WAVE format (commonly known as the WAV-file). The format specification allows for an enormous flexibility but in practice 99% of all WAV-files only contain a header "chunk" giving sample rates, a mono/stereo flag and a few other details and then the actual data chunk with the wave-form. The format can, however, among many other things handle multi-channel sound, loop points and private chunks but almost no playing/editing software can handle it. Indeed, some WAV-file editors and players even fail to read the files at all if they contain anything beyond the basic mono or stereo sound data or, if they can read further chunks, they may not preserve them after a save.

- To enable prediction accurate using sound sources that defy the simple measured fixed 3D-balloon concept, e.g. array or wave-guide designs with distance dependent directivity (non-spherical spreading). In deed, a fixed 3D balloon can easily be 10-20 dB in error for such sources and the reason is not a limited angular or frequency resolution (at 1º and 1/32-octave resolution it will still be 10-20 dB off at some distances).

- To make it possible for a *manuf.* to provide as accurate data as possible and it is up to each *pred.* to use it. It will put some pressure on *manuf.* to make available high-accuracy data or run-time simulation of arrays as well as it puts pressure on *pred.* to use what becomes available. This is in contrast to letting the resolution be dictated by a particular *pred.* because of a 'de facto' standard. However, even with the flexibility the DDI offers, it is still of highest importance to discuss what resolutions are required and the interface may make it simpler to do comparisons between various resolutions.

- To be general and not locked to *CATT-Acoustic*. Any other modeling package running under 32-bit *Windows* can use the specifications of the interface. Interface descriptions will be available in *C*, *Delphi* and *Modula-2* (*Visual Basic* if possible). It is optional for a *pred.* to make its own DLL, e.g. for array modeling or to access its private data-bases, using unique *pred.* features. The benefit would be a unified interface independent of underlying data structures. The interface includes a simple protection mechanism that can be used for private *pred.* DLLs or for those that may be licensed. With *CATT-Acoustic* the *DDI* directivity format is called SD2 in addition to existing SD0 and SD1 formats and is used exactly in the same manner.

- To be upward compatible where one set of features is required and another (possibly expanding) set is optional. As long as the basic features are all implemented correctly old versions of an interface will remain compatible. The *pred.* queries the optional items available and does not call not implemented ones (this typically concerns non-acoustical items e.g. returning a *manuf.* logo or a full 3D description of the loudspeaker cabinet).

- To provide data and "intellectual property" hiding. This is especially important with DSP-controlled arrays where the applied technology may even be patented and the company does not wish to spread their knowledge to every single *pred.* An example of how this can be done is the DURAN_CATT.DLL for *Duran Audio's AXYS Intellivox* series available for *CATT-Acoustic v7.1* where all array specifications and DSP details are completely hidden in the DLL while still making user beam-steering possible.

- To provide a mechanism for on-line documentation of each loudspeaker model implemented via a dedicated WIN32 help-file associated with each DLL. Within this help-file any required model details can be given and if necessary an internet link to the *manuf.* home page for more in-depth information. Further, such a help-file can include step-by-step selection guides and similar for the various models a *manuf.* offers. Since many tools for making HTML-files are available the documentation can as well be made for a www browser or in *Adobe Acrobat* PDF-format.

- To remove the risk of accidental or deliberate altering of directivity data. Also, since the *manuf.* can use his own database, instead of converting to another file format, the risk of errors in translating are diminished and database maintenance is simplified.

- It is anticipated that direct sound and perhaps even 1st order reflections must be handled with higher frequency resolutions than the octave-band resolution currently in use with most *pred.* The interface makes it possible for a *manuf.* to return both 1/1-, 1/3-, 1/6 and 1/12-octave and FIR data (at least 1/1-octave data must be returned). It is then up to each *pred.* to use e.g. FIR or 1/3-octave data for direct sound and 1/3 or 1/1-oct for reflections. Since the interface knows that the FIR and 1/12-octave data is not likely to be called for each single of thousands of rays but only for those that directly hit a receiver, the interface can supply more accurate data that may even be simulated (e.g. for DSP-controlled arrays that are hard to measure because the extended near-field and also because there may a very high number of settings possible). If audience area mapping is used there can be many direct rays but it can be made optional to use the more accurate data since the interface also provides a model-dependent accuracy level control so that a speed/ accuracy trade-off choice can be made by the user or the *pred.*

- The angular resolution of the returned directivity is decided by the *manuf.* and a value is returned for each requested frequency band that typically would be properly interpolated between measurement points or even simulated at the exact angles for DSP-controlled arrays with variable settings (e.g. the DURAN_CATT.DLL). This opens up for use of higher resolution than the

common 10º 1/1-octave format (or the emerging 5º 1/3-octave format) without requiring the specification of a new file format. In deed, the resolution can be 1º in the front (or exact angle if simulated) and 10º in the back since these details are all hidden to the *pred.* Also, a simpler simulation may be employed for array far-field directivity requests (e.g. a fixed pre-calculated or measured 3D-balloon at sufficient resolution). The *manuf.* can do as he means is appropriate for his various lsp models.

- Note: the DDI concept must not be confused with the *Device Driver Interface* of WIN32. Since the WIN32 DDI is known only to hardware driver programmers there should be no problem using the abbreviation also for the current application.

1.3 Interface basics

Since there may be many DLL-interfaces the approach where a reference-file is linked with the *pred.* executable cannot be used but instead all procedures are called using a sequence of :

```
LoadLibrary(…);      (* WIN32: gives a unique BRAND DLL handle *)

GetProcAddress(…);   (* WIN32: gives a pointer to the requested DLL procedure *)
Call the procedure

GetProcAddress(…);
Call the procedure

…

FreeLibrary(…);      (* WIN32: closes the DLL again *)
```

The interface specification provides string literals for public names of procedures and each *pred.* typically writes higher-level wrapper functions of his own choice.

Any *pred.* that wishes to support the DLL-interface needs, in addition to be able to interface to the various procedures defined, needs to provide a user interface to handle, keep track of and allow editing of the following simple items:

- A unique folder for each *manuf.* where the DLL itself and its help-file resides as well as any other data required by the particular *manuf.* The DLL implementor expects to find his data in the same folder as the DLL or in sub-folders organized as the implementor sees fit (the DLL folder can be found by calling the WIN32 `GetModuleFileName()` function). It may sometimes be beneficial to put the data into the resources of the DLL rather than as separate files.

- The DLL name (say `BrandX.DLL`)

- Model names that uniquely identifies each loudspeaker model in the DLL (the interface implements a **`DDI_EnumModels`** function that making it possible to present a model list to a user)

- A version for each model (i.e. which DLL version it is made for)

- Be able to load a WIN32 help-file or display a text-file with dedicated DLL help (with the same name as the DLL i.e. `BrandX.HLP` and located in the same folder). This can be extended to PDF- and HTML-files as well.

- Additional data for those models that have such (e.g. a script that describes an array so that it can be well modeled using maths inside the DLL, or back-panel settings for beam-steering). The data is sent to the interface as a linked list of strings that the interface will know how to interpret. The *Pred.* will only have to maintain the list and make it available for editing.

- The interface implements a model-dependent accuracy setting so that those *manuf.* that wish can offer a choice between speed and accuracy depending on the type of application.

- Depending on how the prediction is made several interface DLLs may have to be open simultaneously each with its own unique DLL handle.

2. Typical calling sequence

(names starting with **DDI**_ and in **Courier Bold** are part of the DDI definition, names in Courier are WIN32 functions)

Get the selected BRAND and model name selected from the user.

Load the BRAND DLL (LoadLibrary)

Initialize the interface (**DDI_InitInterface**)

Initialize the lsp model with the model name and optional data (**DDI_InitLsp**)

Query the DLL for what kinds of resolutions are available for the model (**DDI_GetResolutions**)

Tell the DLL what kinds of resolutions will be used (**DDI_SetResolutions**)

Query the DLL for upper/lower bands available (**DDI_Get MinLower/MaxUpper Band**)

Tell the DLL what upper/lower bands will be used (**DDI_Set Lower/Upper Band**)

If **RES_FIR** is used:

> Query the sample-rates available (**DDI_GetFIRRates**)
>
> Tell the DLL the sample-rate that will be used (**DDI_SetFIRRate**)
>
> Query the FIR sizes available at the sample-rate selected (**DDI_GetFIRSizes**)
>
> Tell the DLL the FIR-size that will be used (**DDI_SetFIRSize**)
>
> Query the valid frequency ranges possible (**DDI_Get MinLower/MaxUpper Frequency**)
>
> Tell the DLL the frequency range that will be used (**DDI_Set Lower/Upper Frequency**)
>
> Query the FIR data-types available (**DDI_GetFIRTypes**)
>
> Tell the DLL the FIR data-type to use: (**DDI_SetFIRType**)

Query the DLL if the directivity is distance dependent (**DDI_IsDistanceDependent**), if it is it should be used by the *Pred.*

Query various other acoustical properties: (**DDI_GetBandSensitivity, DDI_GetBandMaxInputPower, DDI_GetBandDirectivityIndex**)

Query various other items as needed: (**DDI_GetOptions, DDI_GetModelInfoLength, DDI_GetModelInfo, DDI_Get3DOutline**)

Perform the prediction as desired: (**DDI_GetBandDirectivity, DDI_GetFIRDirectivity** (if **RES_FIR** is used))

Release the model (**DDI_ReleaseLsp**)

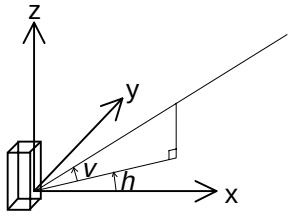
Free the BRAND DLL (release any internal resources at the WEP) (FreeLibrary)

3. Interface definitions

The interface has been designed without using language-specific features so that it can be implemented in any language that can create a WIN32 DLL. In some cases there may be compiler switches necessary to set parameter passing methods but the interface is based on the most common C calling convention for WIN32 (the same as WIN32 itself uses). For a detailed description, contact *CATT* at `catt@netg.se`.

4. Angle conventions

The coordinate system and angle convention used is:



Where

x	is forward (along the geometrical axis)
y	is to the left (as seen from the speaker)
z	is upwards
origin	is at the acoustical center of the source or at a reference point on an array

h	is the horizontal angle 0 to 2π (where 0 is forward and angles increase to the left as seen from the source)
v	is the vertical angle $-\pi/2$ to $\pi/2$ (where 0 is forward, negative angles are above the horizontal plane and positive below).

This convention makes it possible to refer to the on-axis value as (0,0). To convert (h,v) to standard spherical angles (θ,ϕ) (see e.g. Standard Mathematical Tables) use:

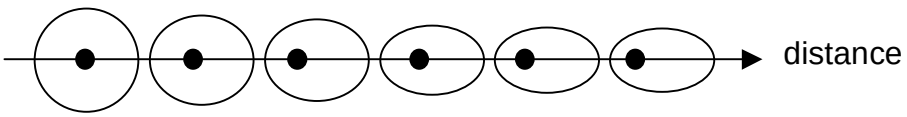
$\theta = h$
$\phi = v + \pi/2$

5. DLL Implementor tips

- Note that even if the number of procedures defined may make it seem like a difficult DLL to create, most of the procedures (functions) answers simple questions (e.g. *What is the highest 1/1-octave-band you can supply?*). Sample template source code will be available.

- For simulated models with variable input data it may be beneficial to cache each calculated value since it may be used again and can be looked up in a list for faster results.

- Depending on the complexity of a loudspeaker or array, any approach from fixed balloons, to distance dependent balloons (interpolated between balloons based on distance, see fig) to full-blown simulation can be hidden in the DLL interface. The fig illustrates an approach with pre-calculated or measured balloons at several distances that are interpolated in-between:



The `CATT_Generic` DLL, in addition to array modeling, offers a model that takes a number of measured or simulated full-space 10º balloons at a number of distances and interpolates between

them automatically. This can be a way to quickly get distance-dependent data into a useable format before deciding on making a dedicated DLL and is nearly as fast as using a normal fixed 3D balloon.

- Data can be put in a resource file and bound to the DLL rather than put as separate files. The benefit is data hiding and easier maintenance since only a single DLL file is required.
