

PRESAGIS

Mastering the ARINC 661 Standard

By Yannick Lefebvre

Table of Contents

Abstract	3
Introduction	4
ARINC 661 Standard Overview	5
Architecture Overview	5
The Cockpit Display System	6
Layers	7
Standard Widget Library	8
User Applications	11
Runtime Protocol Definition	12
Distributed Development Benefits	13
Goals of ARINC 661	13
Certification.....	13
Practical Real-world Considerations	15
Implementing a subset of the standard	15
Bandwidth and Performance	15
Interoperability	15
Design Updates.....	16
Using commercial tools to address the standard	17
Definition File Creation.....	17
Creation of binary definition files.....	17
Widget Library Creation	17
CDS Simulation	18
Embeddable CDS Kernel	19
Document Generation.....	19
Support for new ARINC 661 features.....	19
VAPS XT 661	20
Summary	21
About the Author	22
About Presagis	23
References and Additional Information	24

ABSTRACT

By introducing a separation between graphics and logic, an interpreted runtime architecture and a defined communication protocol, the ARINC 661 Standard was designed to address many of the concerns that aircraft manufacturers face today when creating cockpit avionics displays. Before starting a project using this technology, it is important to understand all of the elements introduced by the standard, along with the benefits and occasional drawbacks.

This workshop will first provide an overview of ARINC 661 concepts, then go on to describe the benefits of using a distributed development approach as well as cover some practical real-world considerations when implementing an ARINC 661 solution. Finally, we will see how commercial tools can be used to simplify the creation of displays following this standard.

INTRODUCTION

The task of creating aircraft cockpit displays has grown increasingly difficult over the past decade due to certifications rules becoming stricter and being applied more widely (DO-178B) along with the constant drive for faster to market with lower development costs. To make things even more complicated, every player in the industry have had their own development methodologies with little to no guidelines on contents other than the opinion of their developers and human factors engineers.

In the late '90s, a committee of representatives from the industry was formed to address these concerns. This committee was very representative of the marketplace with members coming from aircraft manufacturers, hardware suppliers as well as software vendors such as Airbus, Boeing, Rockwell Collins and Presagis. The first version of their work was published in 2001 by ARINC under the title "Cockpit Display System Interfaces to User Systems, ARINC Specification 661" and was used to develop displays for the Airbus A380. Subsequently, four supplements were created to clarify and augment sections of the original document based on lessons learned during the development of the A380 and to include additions proposed when new members joined the committee. At the time of this writing, a fifth supplement is underway and should be available in the near future. After the A380, ARINC 661 has been selected to develop displays for the Boeing 787 and the Airbus A400M, as well as the upgrade to the AgustaWestland Merlin helicopter.

From a technical perspective, the ARINC 661 Specification defines an overall architecture along with many sub-components to facilitate the creation of interactive displays. The first of these components is the Cockpit Display System (CDS), a rendering engine dedicated to presenting graphical information. Of course, a display system would not be much without its associated logic which is handled by a separate element called the User Application (UA). The link between these two pillars is the ARINC 661 Runtime Protocol, which carries events that are generated through user interaction to the UA and brings requests to display new data back to the CDS. Finally, the contents of the displays, from graphical elements to possible groupings of these components, are defined by using a finite set of components called the widget library.

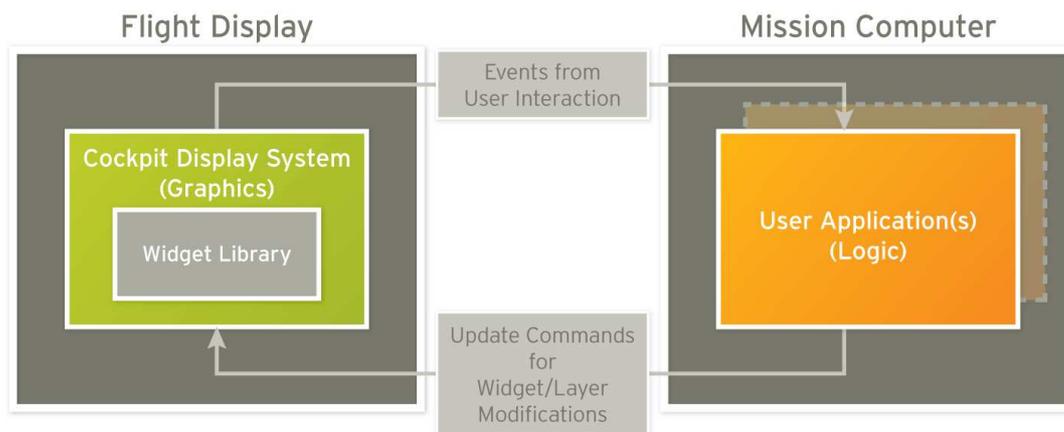
ARINC 661 STANDARD OVERVIEW

ARCHITECTURE OVERVIEW

While cockpit display software has traditionally been written as self-contained executables that present information based on internal rules and logic, ARINC 661 introduces a clear separation between the code drawing the graphics and the code managing the logic, position and state of all of the visual elements. These two components are called the Cockpit Display System (CDS) and the User Application (UA). Furthermore, ARINC 661 defines the CDS as a runtime interpreter capable of displaying one or more elements from a finite library of building blocks called widgets.

After making these two application components independent, the next step was to define a standard for them to exchange messages. Using the ARINC 661 Runtime Protocol, messages are carried from the CDS to one or more UAs when user interaction occurs. Once messages are processed, the UAs send messages back to request updates and changes to the widgets displayed by the CDS.

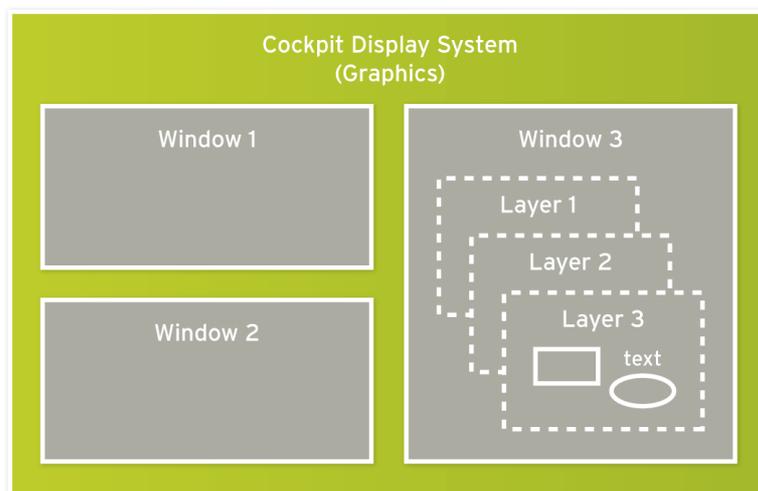
With all of these elements in place, the high-level system architecture of an ARINC 661-based flight display follows the diagram shown below.



THE COCKPIT DISPLAY SYSTEM

The Cockpit Display System is responsible for displaying widgets to the end user by using a library of all the widgets defined by the ARINC 661 specification. At startup, the CDS loads and displays widgets based on one or more binary file(s) called Definition File(s) (DFs). Each Definition File contains one or more layers, which are hierarchical listings of all of the widgets that need to be loaded, once or multiple times, along with their initial properties, such as position, color and visibility.

Going down a level, the physical display attached to the CDS is divided into one or more subsections, simply called windows, which can render one or more layers. These windows cannot have any overlaps and will stack the designated layers to create the final result.



Once the CDS has completed its loading phase, all changes to the properties of the widgets are controlled by the User Application through the ARINC 661 runtime commands. During execution, the CDS is responsible for handling events that come from cursors, touch screens, keyboards or other input devices. These interactions can result in a change to the widget's visual appearance (for example, the CheckButton highlights when the cursor is located over the widget) or in the generation of an event due to a selection being made. The UA only receives these events and is not aware of specific user interactions.

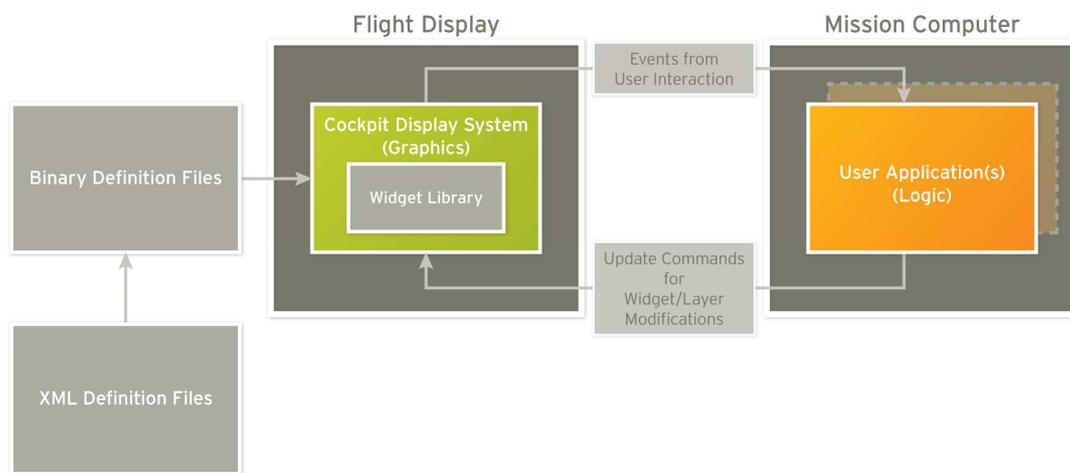
There are many benefits to this runtime interpreter approach. The first of these is that the code of the CDS only needs to be written, compiled, tested and certified once. After the interpreter is finalized, updates and changes to the composition of the display are done by creating a new Definition File that will be loaded in the CDS. The same benefit applies to changes to the logic flow of the application which only result in changes to the user application.

In addition to this, the decoupling of interests facilitates the division of tasks between the graphic programmers and those that need to manage the logic and other aspects of the display system.

LAYERS

Layers define groups of widgets along with their initial position and properties. All of these widgets are defined in the widget library. While widgets are given names in the definition files, it is really their designated IDs that will be used to identify them while a display is being executed inside of the CDS. This type of identification simplifies the exchange of messages between the UA and the CDS. To avoid conflicts, all widgets within a layer have distinct widget IDs. However, different widgets in different layers can have the same ID since the layer ID is also used in runtime messages to uniquely identify a widget.

Layers must be saved as binary files to be loaded inside of the CDS. However, for practical editing purposes, layers are usually stored in an XML format that is also defined in the ARINC 661 specification. Once the content creation is complete, the textual XML files are converted into binary files.



Besides being a simple recipient, layers also have runtime properties that can be modified such as their visibility and activation state. When a layer is invisible, all of the contents will not be seen on-screen. When a layer is inactive, none of the widgets that it contains will be selectable.

STANDARD WIDGET LIBRARY

The basic building blocks of a Definition File that will be displayed in a Cockpit Display System are simply called Widgets. The first version of the ARINC 661 Specification introduced 42 widgets that could be used to create displays. This number went up to 50 with the first update to the standard, to 57 with supplement 2, to 65 in revision 3 and to 68 with its most recent incarnation published earlier this year.

Having a standard set of widgets to develop a display makes it very easy for a developer to become familiar with the ARINC 661 standard and to understand how to develop new displays. This standard set of building blocks also gives flexibility for a content developer to use CDS systems from multiple vendors as they see fit since the definition files follow a standard format. Finally, it is also possible to re-use large parts of a definition file on a new project by modifying the visual appearance of widgets to create a new CDS library.

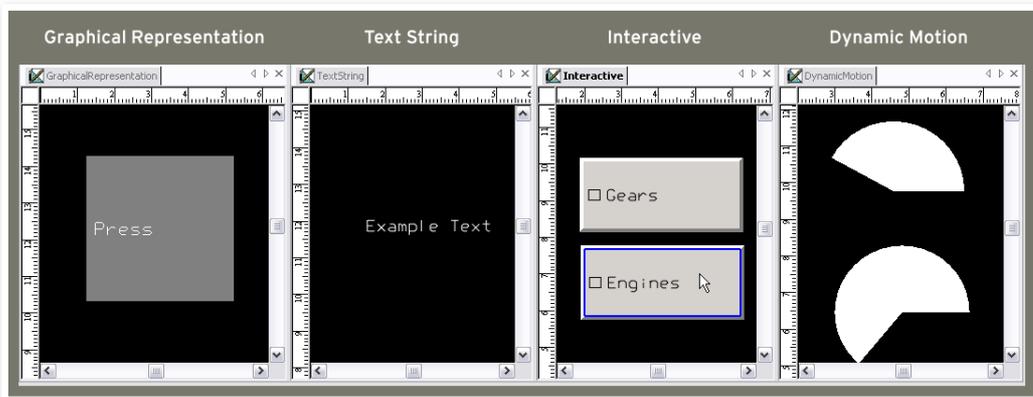
Widgets vary in complexity from basic graphical elements such as the GpLine and GpRectangle widgets to complex objects such as the MapHorz widget which displays maps from various data sources. There are also some widgets that do not have any visual representation and are used to group other elements as well as apply transformations to them. An example in this last category is the MutuallyExclusiveContainer widget that groups multiple elements but also only displays one of its immediate children at a time.

Widget Categories

In total, there are eight categories that a widget can belong to. Here is a complete list of these categories along with examples of each type:

- Container: A widget that can contain other widgets to define hierarchical organization or to provide functionality (e.g. MutuallyExclusiveContainer)
- Graphical Representation: A widget that draws graphics (e.g. PushButton)
- Text String: A widget that renders text (e.g. PushButton)
- Interactive: A widget that reacts to interaction from the user (e.g. PushButton)
- Map Management: A widget related to the rendering of a map inside of an ARINC 661 display (e.g. MapHorz)
- Dynamic Motion: A widget that can be repositioned while the display is running (e.g. GpLine)
- Utility: Widgets that are not containers, do not have graphical representation and that are not interactive. These widgets have specific functionality in order to extend or optimize the ARINC 661 defined principles (e.g. BufferFormat)
- UA Validation: Widgets which may have their events (if applicable) validated by the UA (e.g. XXXXXX)

As we can see from these examples, a widget can belong to more than one category based on its functionality.



Widget Definition

One of the primary purposes of the ARINC 661 Specification document is to provide information on all widgets. This results in two-thirds of the document being dedicated to the Widget Library.

Each widget definition in this library contains a series of standard sections that define the widget in detail. These sections are classification data, properties, Creation Structure, and Widget Description.

- Classification Data

Indicates which category(ies) a widget belongs to.

- Description

This section provides a textual description of the widget functionality. While this description gives a good general idea of the functionality of the widget, it does not provide any information on its visual appearance. This leaves the task of defining the look and feel of a widget completely up to the CDS developers.

- Special Comments

Contains high-level observations on the widget's functionality and identified relationships between multiple widgets found in the standard,

- Restrictions

Identified widgets that can only be used in special conditions or that have properties that can only be configured a specific way.

- Parameters

Besides the obvious name and definition of each property, this table also contains a column indicating whether the widget can be altered at design time, runtime, or in both modes. For most widgets, only a small number of their properties can be updated during the execution of the display. For example, if we look at a widget such as the PushButton, we can see that only four of its properties can be modified: Visibility status, Enable flag, StyleSet property and its Label text. While this type of restriction might seem like it would make displays very static, other widgets such as the TranslationContainer (a grouping widget that can move along with its children), can be used to move elements around on the display.

Besides a few exceptions, such as ColorIndex property of a GpLine, the majority of the widget properties are not used to define the appearance of a widget instance. Instead, they focus on specifying its location on the display as well as its initial settings and behavior.

- Creation Structure

This section has all of the same property names as before but goes into more technical details about them including their data types, size in bits and valid range of values. It is interesting to note that the order in which these properties are organized is not the same as the previous table. The reason behind this is a rule of the ARINC 661 specification that states that all properties must be arranged in groups totaling 32 bits, all the way to the end of the list.

This rule was created to simplify the alignment of data inside of the binary Definition Files when creating layers of widgets and to simplify memory allocation inside of the CDS. However, satisfying this need means that unused properties must be added in some cases to get the correct alignment.

- Event Structures and Runtime Modifiable Parameters

Contains a list of events that the widget can generate while the application is running and information on the messages that need to be sent to assign new values to the runtime modifiable parameters.

Style Sets

To achieve a unified look and feel across a display, ARINC 661 specifies that the look and feel of widgets is defined inside of the CDS. This means that all of the widgets of a certain type will look the same when used inside of a display. However, ARINC 661 also defines a property called the StyleSet to allow a little flexibility in configuring the appearance of objects.

The StyleSet property is a numeric field that can be used at the discretion of the CDS developers. In many implementations of ARINC 661, the StyleSet has usually been used as an index to identify one of many graphical representations of a widget that should be displayed. For example, the StyleSet of a GpLine widget could be used to select its line pattern. In other instances, developers wanting more control over the visual aspect of their widgets used the StyleSet as a long sequence of numbers, also known as a bit field, where each element has its own signification. For the GpLine widget, the same field could be used to indicate line thickness, pattern and opacity using this method.

Custom Widgets

For some projects, the set of widgets contained in the ARINC 661 standard may not be large enough to implement the desired functionality. In these cases, the standard opens up the possibility of creating custom widgets. Custom widgets will still have some common properties that all widgets have but will then be able to have a tailored list of fields to configure it at design time and to be able to send data during execution.

Custom widgets are also able to use the standard ARINC 661 runtime protocol and all other aspects of the standard.

USER APPLICATIONS

As mentioned previously, the ARINC 661 architecture introduces a separation of logic and graphics. The user application (UA) is responsible for providing data to update the contents of the CDS based on flight data and to react to user interactions with the display.

The ARINC 661 standard does not go into great detail in terms of UA architecture. The first statement made is that a UA can be connected to one or more layers that are loaded in the CDS. The specification also specifies that communications between the CDS and the UA must use the ARINC 661 runtime protocol. Besides these two basic points, the User Application can be created using any programming language or design methodology.

First and foremost, the benefit of separating graphics and logic is that a modification to the logic will not impact the certification of the CDS. This approach also facilitates the distributed development of a cockpit display across multiple teams and facilitates the simulation and testing of displays.

RUNTIME PROTOCOL DEFINITION

The ARINC 661 specification defines a bi-directional protocol that is used by the CDS and the UA to exchange data and events. The definition of this protocol is made possible by the set of standard widgets discussed earlier and the architecture put in place by the standard.

Just like the properties of the widgets, data contained in runtime protocol commands must follow a very specific order to maintain the data alignment rules specified by ARINC 661. In most cases this means creating unused variables to properly align data in memory. One interesting point concerning runtime commands is that layers and widgets are referenced using the IDs that they were assigned in the Layer definition. Therefore, all information that is transmitted inside of these messages is numeric in nature.

While the contents of the data packets is very precisely defined, ARINC 661 does not mention the data transport mechanism to be used to carry messages. This allows companies to use a protocol that is either specified in the contract or with which they have the largest experience base.

Execution flow

When a cockpit display system built using the ARINC 661 is launched, the contents of the display are first loaded from the binary definition files that are contained in the device's local storage. All of the widgets listed in the files are loaded from a library contained inside of the CDS and configured according to the properties listed.

Once all DFs are loaded, the system transitions to the Runtime Phase where the UA becomes the main controller of the contents of the CDS. The UA can send commands to request updates of the runtime-modifiable properties of widgets or to ask the CDS to modify properties of a layer such as visibility. Messages are typically only sent between the two applications when changes are needed. This results in a lower use of the processor to handle the network traffic and avoids unnecessary rendering of the display.

DISTRIBUTED DEVELOPMENT BENEFITS

GOALS OF ARINC 661

From the early stages of its creation, the ARINC 661 standard was designed to meet a series of guiding principles, namely:

- Minimize costs for new display capabilities
- Minimize costs for changes to new display capabilities
- Minimize costs for the management of hardware obsolescence
- Support the means of developing standard HMI

Looking back at the architecture defined by ARINC 661, we can see that many of these goals are met through the basic architectural design of separating graphics from logic. This isolation simplifies the design and implementation of both components while the precisely defined protocol message format ensures that integration will go smoothly between the two main application components. This split design also simplifies upgrades or unit replacements since a new component can be quickly dropped in and will be compliant to the standard.

Another important benefit derived from the separation between the CDS and UA is the ability to easily distribute the development of these systems to independent software development teams that can work in parallel, thus shortening the entire process. It also gives more liberty to the user interface designers to experiment with display concepts and make changes quickly without needing to recompile code. All they need to do is save a new binary definition file and they can rapidly see the results of their work.

Moving on to the aspect of cost reductions for new display development, while the creation of a first CDS does involve a fair amount of upfront development time, like most new avionics developments, additional instances of a CDS can quickly be deployed on additional systems with a new set of layers to put new displays in place. The same applies when an entirely new airframe project gets underway where a large part of the CDS be re-used, with potentially only some aesthetic modifications to the widget library to put in place the desired look-and-feel for the new flight deck.

Finally, the use of a standard widget set across all flight decks system will ensure that all displays have a standard look-and-feel and common elements that the crew will quickly become familiar with.

CERTIFICATION

On most airborne vehicle displays, compliance with DO-178b (or equivalent) development methodologies must be followed, even in cases where the project is not actually presented to the certification authorities. ARINC 661 excels once again in this area with its segregated design and well-defined architecture. The separation of graphics and logic means that a change to an initial widget property or to a static widget property can be done by simply changing the contents of a definition file without affecting a single line of code. Other changes such as the update to a widget's appearance or a change to the display logic are once again isolated to a single system and only require re-certification of that individual system. The accelerated development benefits mentioned in the previous section also apply to certification where re-deploying a certified CDS to a new project can be done with minimal work.

Another benefit of ARINC 661 that is often overlooked is how the very precise descriptions of all of its components can be used as source materials for the high and low-level requirements of a display when preparing all of the required certification documents. More specifically, the top-level CDS / UA / Runtime Protocol architectural component design can be used as part of the system's high-level requirements while more precise elements such as the details of each widgets properties would become part of the system's low-level requirements.

Finally, pairs of layers and UAs could technically have different levels of certification. Of course, since the CDS is a single executable, its code needs to be certified to the highest level required for a system, but the UAs for low-criticality systems could be certified at a lower level to avoid some of the costs of higher levels.

PRACTICAL REAL-WORLD CONSIDERATIONS

As we have seen through this document, ARINC 661 brings a lot of great concepts to the table for display development. However, some aspects of this architecture are not always as practical as they might seem. The following section looks at some considerations to be taken when selecting this standard.

IMPLEMENTING A SUBSET OF THE STANDARD

While the standard defines a large number of widgets, trying to implement every last one of them could be a very time-consuming effort. A more realistic approach would be to do an up-front analysis of the widgets that will be required to implement an aircraft's layers and only implement this subset. Similarly, runtime messaging commands can be simplified by only addressing the widgets that are required for a specific display system.

BANDWIDTH AND PERFORMANCE

The precise and complete qualities of the ARINC 661 runtime communications protocol also mean that these messages are quite large and have some overhead when compared with traditional small and compact ICD messages.

For example, in traditional ICD communication schemes, data to drive display elements such as a heading, a single readout and a compass orientation could all be transmitted in a compact 16-bit data packet. The equivalent ARINC 661 commands to set widget values to represent these three values would need to be split up into individual messages that each weigh in at 64 bits, representing twelve times the amount of data to send equivalent information.

The bandwidth issue becomes even more evident for displays that have frequently changing elements since bus throughput will quickly become a limiting factor to the refresh rate of a display. If we expand the previous example a bit more, a display screen that is built from 30 dynamic elements would require 30 set parameter commands to be set to update all of its elements. With each message weighing in at 64 bits, and overhead control message data adding an extra 96 bits of information, we are looking at a total of 2016 bits of information to refresh the entire display. For a bus system like ARINC 429, this amount of data puts the maximum theoretical refresh rate at 49 hz. Of course, this is in an optimal situation where no other data needs to share the same communication bus.

Another aspect of bandwidth to be aware of is that UA response might be slowed down if there is already a lot of data traffic on the bus being sent to update a display, which might cause a system to quickly become unusable.

The main way to avoid this issue is to be aware of the problem and to avoid sending updates to all components unless it is absolutely necessary. Some components might only need to receive updates at half the frequency of more critical elements. It is also critical to have a good understanding of the performance of the communication system early in the project to avoid costly redesigns down the road.

INTEROPERABILITY

While the concept of interoperability was very present in the initial version of the ARINC 661 standard, this interoperability has been somewhat diminished by the subsequent release of the

4 supplements since each one introduces new widgets that older CDS systems will not support. Each addition also makes slight changes to existing widget definitions and to the interactions between the CDS and the UA. Finally, the provision for custom widgets directly implies that a CDS from one vendor can have some functional elements that another CDS will not support.

Putting extra care in the design of the UA can help alleviate some of these issues but there will likely always been some incompatibilities between a CDS and a UA developed for different projects or by different vendors.

DESIGN UPDATES

As covered in the section on window and layer positioning, multiple layers can be loaded in a window and can potentially overlap each other. This means that extra care needs to be had when making changes to the content of layer that will be displayed in such a shared environment to avoid obstructing elements from another layer when an update takes place. There also needs to be some thoughts towards avoiding a drop in performance or spikes in network traffic when adding new elements to an existing system, since new elements will translate into additional rendering time and additional network messages.

USING COMMERCIAL TOOLS TO ADDRESS THE STANDARD

DEFINITION FILE CREATION

The first concern when creating ARINC 661 displays is creating the definition files that describe the contents and layout of each layer that will be displayed in the system. While ARINC 661 clearly defines the XML format for definition files, manually editing such files is prone to human error.

ARINC 661-based development tools simplify the creation of definition files through easy-to-use interfaces and direct preview of the resulting displays. Some more concrete benefits include the ability to align and distribute graphical widgets, to easily copy one or more elements without having to manipulate XML and a direct feedback during color selection to create visually pleasing displays.

As displays are created using a development tool, definition files are usually directly stored in the XML format defined by the standard so that they can easily be converted to the binary format required by the CDS. Additional benefits of storing files in an XML format is the ability to track changes through a version control system as well as the ability to view the data in human-readable form.



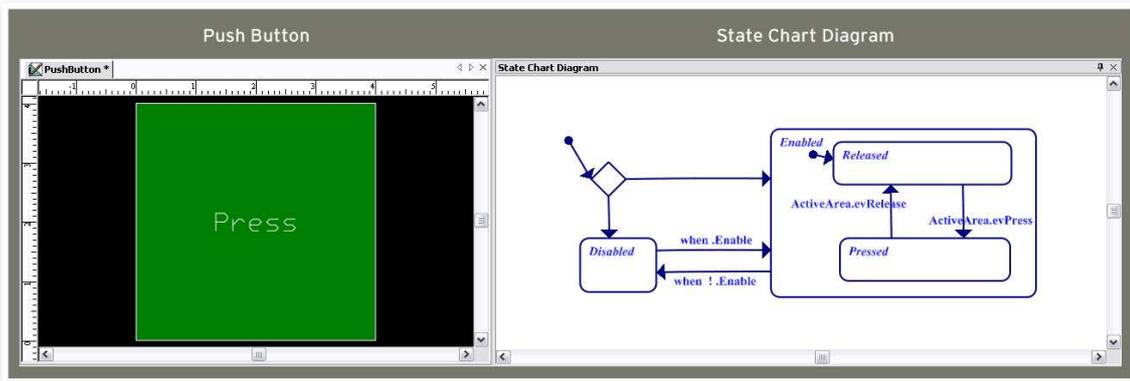
CREATION OF BINARY DEFINITION FILES

The natural follow-on step is the ability to convert the XML definition files to the binary format that is required by the CDS for execution. The automatic generation of binary DF files saves a lot of manual work for developers and avoids conversion errors.

WIDGET LIBRARY CREATION

Tools supporting ARINC 661 development can also be used to develop the look and feel of a complete widget library without needing to work on the actual CDS hardware. These widgets can be created graphically, using the low-level building blocks that are provided with the tools, or through programming.

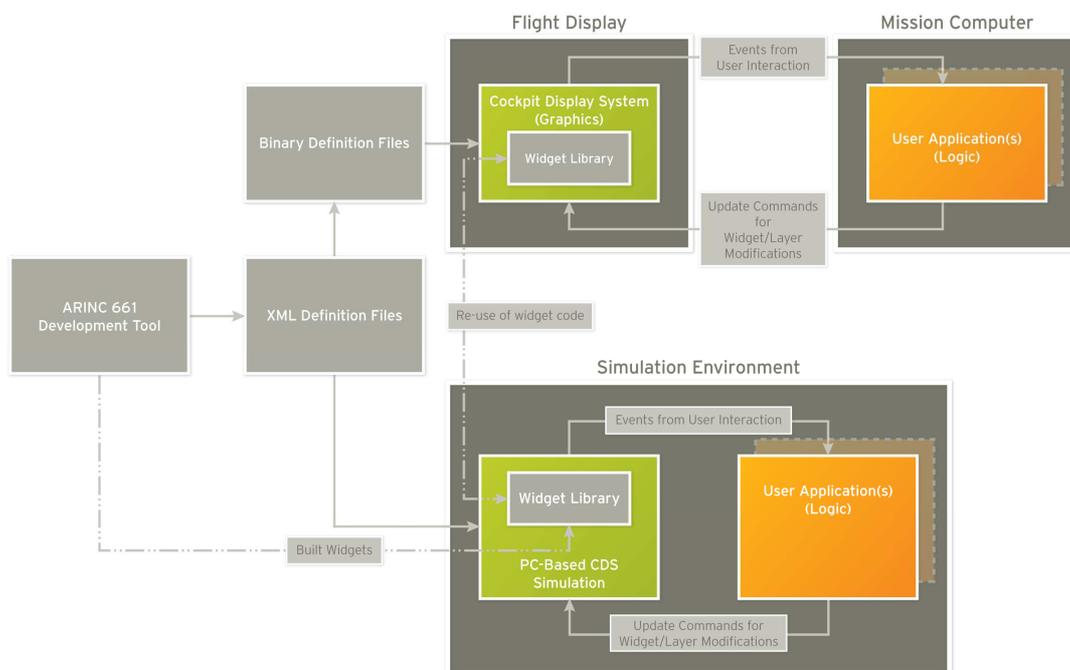
Creating widgets graphically enables quick changes to be made to their appearance during an iterative development process without needing to modify or create a single line of code. Once the graphics are defined, integrated state chart logic and data flow connections can be used to model the desired behavior.



Alternatively, widgets can be created programmatically and integrated in an ARINC 661 development tool through a plug-in architecture. While the technical knowledge required to create widgets this way is a little higher than creating built widgets, this method facilitates reuse of code between the development environment and the final CDS code. Using coded widgets also allows testing of the widget code to be done on a low-cost PC platform instead of running all of the test suites on the costly CDS hardware.

CDS SIMULATION

Finally, CDS vendors often focus on the creation of the CDS software that will run on their embedded solution and don't always stop to think about giving their customers or internal developers the ability to simulate their layers in a desktop environment. Windows-based commercial tools provide all of the tools to let developers get an interactive preview of their application and see the results of their selections, complete with simulation of ARINC 661 runtime protocol messages.



EMBEDDABLE CDS KERNEL

The next large burden that needs to be carried by software developers is the creation of the embedded CDS software. The solution to this is to use a commercial-off-the-shelf embeddable CDS kernel.

Through a customization process, the code of the CDS kernel is able to run on any combination of hardware, operating systems and graphic libraries to render the customer's widget library and definition files. Another related improvement is the ability to code-generate widgets constructed graphically through tools to incorporate them in this embeddable CDS framework.

Of course, an important point when using a commercial solution is to be sure that it supports the DO-178 certification standard since all of the code running in commercial airframes needs to adhere to this when used in safety-critical environments.

DOCUMENT GENERATION

One of the most daunting tasks in display content creating is to document the work done. To address this, automatic document generators can take the displays created and automatically generate specifications and design documents from definition files and widgets created in the tool.

SUPPORT FOR NEW ARINC 661 FEATURES

Finally, since ARINC 661 is an evolving standard, new features are constantly added to support functionality that gets introduced with each new supplement.

VAPS XT 661

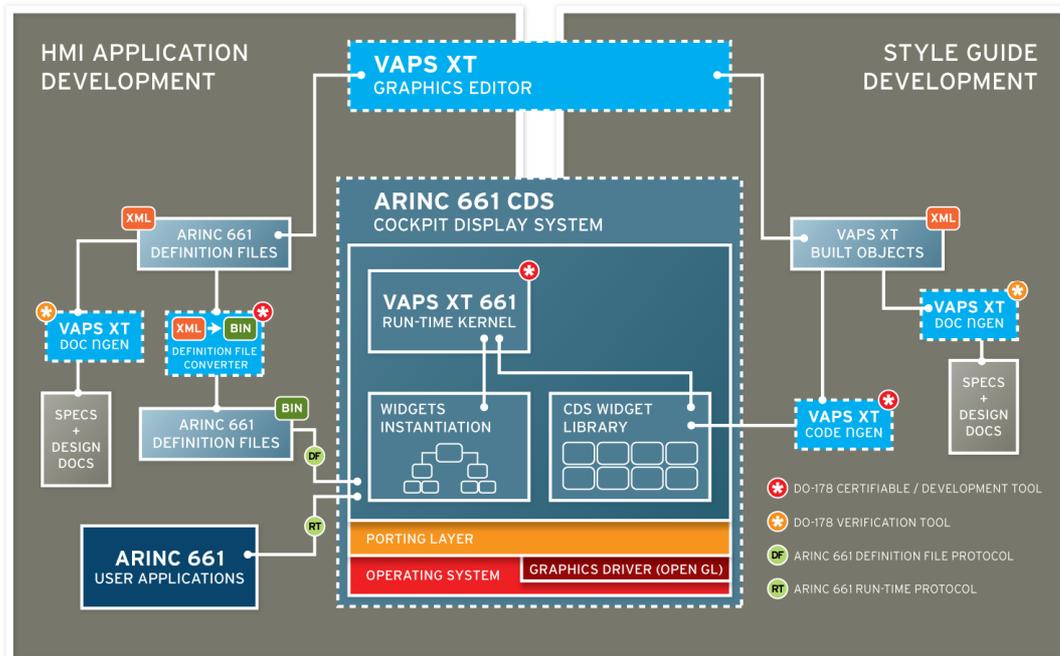
The VAPS XT 661 tool suite from Presagis simplifies all aspects of ARINC 661 display creation. Since its introduction in 2004, it has been used to create content on multiple programs around the world.

With time, VAPS XT 661 has grown to support the entire ARINC 661 development cycle, as can be seen from the list of features:

- A full-featured graphics editor to create layers and definition files, define and simulate widgets as well as export binary DF files
- An embeddable CDS kernel that can be executed on any target through the use of a porting layer
- A code-generator to bring graphically-created contents into the embedded CDS
- A document generator to produce specifications and design documents from all of the definition files and widgets created
- A path to DO-178 certification for all code that will be running on the embedded target and for the tools used in the process

The diagram below shows how all of these components work together to provide a complete ARINC 661 development workflow.

VAPS XT ARINC 661 ARCHITECTURE



SUMMARY

ARINC 661 is a very exciting technology that introduces many benefits along with an interesting architectural design. Within the current climate of faster time to market and reduced development cost, development re-use combined with the stability of commercial solutions is a winning combination. After seeing a few large commercial aircrafts lead the way, many programs are considering or have already adopted ARINC 661 for their upcoming projects, ensuring the success of this standard.

ABOUT THE AUTHOR

Yannick Lefebvre is Technical Marketing Manager at Presagis with a background in computer sciences. Capitalizing on fourteen years of experience in the training, support, sales and marketing, Yannick has provided counsel on hundreds of simulation and embedded display programs globally and is considered an expert in the HMI tools industry. Questions and comments can be directed to: yannick.lefebvre@presagis.com

ABOUT PRESAGIS

Presagis is a world leader in commercial-off-the-shelf (COTS) modelling, simulation, and embedded display graphics software, and creator of the first unified COTS modelling and simulation software portfolio for the global aerospace and defence markets. As an independent, wholly-owned subsidiary of CAE, Presagis is built on a combined legacy of innovation and service, delivering superior customer value through rapid technology advancement. The company services more than 1,000 active customers worldwide, including many of the world's most respected organizations such as Boeing, Lockheed Martin, Airbus, BAE Systems and CAE.

REFERENCES AND ADDITIONAL INFORMATION

1- Wikipedia.org, ARINC 661 Dates and information

http://en.wikipedia.org/wiki/Arinc_661

2- ARINC 661 COCKPIT DISPLAY SYSTEM INTERFACES TO USER SYSTEMS
ARINC SPECIFICATION 661-4