International Color Consortium®

White Paper #52

Level: Advanced

Date: June 2020

iccMAX calculatorElement Security Implementation Notes

A guide for implementing secure calculator element processing

1 Background

Color management with iccMAX profiles defined by either the ISO 20677 or ICC.2 specifications (or Interoperability Conformance Specifications in relation to one of these specifications) utilizes customizable sequences of processing elements to encode color transforms between device color encodings and device independent color encodings. (Note: This provides extensive flexibility over the limited and fixed sequences of transforms utilized by profiles defined by either ISO 15647 or ICC.1 specifications).

The processing elements that can be encoded as part of a MultipleProcessingElementsType based tag provide data for code within an implementing Color Management Module (CMM) to perform the transform implied by each processing element. Most of these transformations have fixed behavior (like applying a matrix, applying a specific function relative to a fixed set of constants, or performing interpolation in relationship to a lookup table).

With ICC.1 based transforms the only way to encode multi-channel non-linear functions is to utilize a multi-dimensional lookup table. In this case the non- linear functions are only approximately reproduced (due to interpolation) and high dimensionality of input channels results in either very large or sparsely sampled lookup tables (limiting accuracy).

When contemplating the functionality required by more complex color management scenarios it was felt that the need to directly encode functions and limited algorithms would be beneficial as the iccMAX specification was developed. Thus, considerable effort was employed to provide more flexible direct encoding of functions and algorithms within a Calculator processing element. Two key aspects that are very significant in the design of the Calculator processing element are security and predictability.

The purpose of this paper is to provide those that both implement and utilize CMM's that support application of iccMAX profiles containing calculator processing elements an understanding of the security risks involved and implementation

guidelines to mitigate such risks and provide predictable results. Three aspects of a calculator element are explored: parsing, validation, and application.

2 Parsing the calculatorElement

The single biggest aspect of parsing a calculatorElement is ensuring that the implied structures conform to what is required by the iccMAX specification, and aborting the parsing if encoding conformance is not achieved. Implementations that fail to ensure conformance to the structures defined by the specification will be subject to various security risks due to possible memory corruption as part of the parsing process.

From a high-level perspective, the calculatorElement is encoded similarly to all other processing elements with a header followed by data for the processing element (see figure 1). Note: The exact byte encoding for the calculatorElement is specified by both ISO 20677 and ICC.2 and the interested reader should consult the specification for exact details.

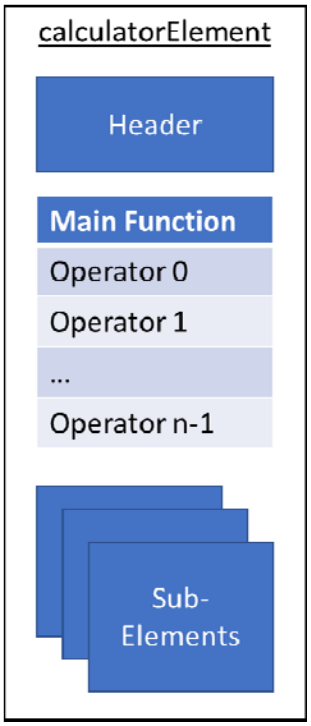

Figure 1- High-level view of calculator element encoding

The header size is defined by the iccMAX specification and contains the standard process element header values (i.e. the process element signature ('calc' for the calculator), the number of channels going into the element, the number of channels going out of the element) as well as references to the data used by the calculatorElement. Data within a calculatorElement is made up of a main function (made up of an array of script operators - required) and an array of additional sub-elements (optional - as referred to by operators in the main function).

The main function as well as each of the sub-elements are referenced in the header using a positionNumber which provides both relative offset and size of data associated with the object that the positionNumber references. Encoding of the main function is also defined by the iccMAX specification which contains a signature,

number of operators followed by the actual an array of operations. Each operation contains a 4byte signature and 4 bytes of constant operand data (the meaning and use of the operand data depends on the nature of the operation defined by the signature). Parsing the main function is implementation specific, but can be as simple as allocating a byte array with the size of 8 times the number of operators, and reading in the whole function array into the byte array. Checks on allocation failures should be made before reading any data from the profile to avoid corrupt memory.

Each of the sub-elements of a calculatorElement should also be successfully parsed and with kept track of as part of the calculatorElement memory storage. If any sub-element cannot be successfully parsed, then the parsing of the calculatorElement should be aborted with an error condition.

When parsing data objects (like the header, main function or sub-elements) within a calculatorElement the offset and size associated with the object should be checked so that it fits within the containing object before reading the object's data from the profile file (as should be done for all objects parsed from an iccMAX profile). This avoids complications such as memory overrides or data corruption when parsing objects. Implementations that fail to ensure that data lies within its containing object will be subject to various security risks.

It is recommended that parsing be aborted with an error condition when the indicated data for any object is found to lie outside of its containing object.

3 calculatorElement validation

A calculatorElement is valid for application when both its main function is valid and all the sub-elements of are valid.

Validation of the contents of the main function of a calculatorElement involves checking to make sure that the future application of the main function by the CMM will be safe and secure. This can be performed either as part of parsing the data for the element or as part of a separate step evaluated before calculatorElement application is performed. **Note: The iccMAX specification requires that calculatorElement validation be performed as part of a conforming consumer of profiles.**

Validation of a calculatorElement is performed relative to the various participants in calculatorElement application. The runtime participants of calculatorElement application are shown in Figure 2.

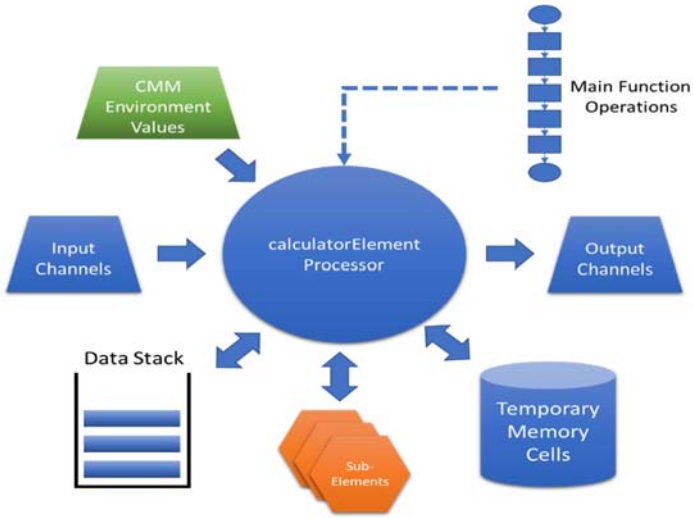

Figure 2 – Runtime participants in calculatorElement application

At the point when application of the main function of a calculatorElement is performed it involves applying operations from the main function one at a time until the last operation is applied. Each operation in the main function has a 4-byte signature and 4-byte operand data used to control how the operation is applied.

Nearly all operations are performed relative to placing, manipulating, or removing data from an operation data stack. They may also involve getting data from the input channels, putting results to the output channels, getting data from a CMM environment variable, getting/putting data from/to temporary memory cells, and invoking sub-elements.

In order to ensure the secure operation of calculatorElement it is critically important that the operations of the main function are validated to: ensure that all operators are supported by the implementing CMM, ensure valid input/output channel access, ensure valid temporary memory access, ensure valid operation stack use, and ensure valid addressing of sub-elements. If there is a failure in any of these validation aspects then application of the calculatorElement should not be performed.

A design-criteria of the calculatorElement is that run-time behavior validation can be performed without having to apply its main function. This is made possible by the fact that the control of both size and position of data access by operators is statically determined only by the operand data provided with each operator in the main function. Note: The encoding and use of operand data is specified for each calculatorElement operator in the ISO 20677 or ICC.2 specification.

Validation of a calculatorElement main function can be performed in three parts: operation validation, branch validation, and validation of operation stack usage. Whether these are performed separately or together in some fashion is determined by the implementer.

3.1 Operation Validation

Operation validation involves checking each operation in the main function and first determining if it has a supported operator (defined by the operator signature), then checking that the operation is valid in the context that it is used, and finally

determining if the operation utilizes an out of bounds access (defined by the control operand(s) in the data part of the operation).

Operator Support

Supported operations are those that have an operator signature for operations that the CMM had the ability to apply. Some operations (like '***env*** ' and '***solv'***) may not have internal support by the CMM but are still supported as valid operators by the CMM with expected behavior (as defined by the iccMAX specification). If the signature of the operation isn't in the list of supported operations of the CMM then both the operation is not valid as well as the containing calculatorElement, and application of the calculatorElement should not be performed. Note: All operations of a calculatorElement main function should be evaluated for operator support regardless of the branching context that the operation has in the main function.

Context Checking

A few operations defined by the iccMAX specification are only valid in specific contexts because they are used as operation extensions.

The '***if*** ' operation can be extended by the '***else'*** operation and therefore the '***else'*** operation is only valid if it is proceeded by an '***if*** ' operation.

The '***sel*** ' operation is extended by the '***case'*** operation and therefore the '***case'*** operation is only valid if it is proceeded by a '***sel*** ' operation.

The '***sel'*** operation can also be extended by a '***dflt'*** operation and therefore the '***dflt'*** operation is only valid if it is proceeded by a '***case'*** operation.

Operation Bounds Checking

Operations involving input/output channels, temporary memory cells, or sub-elements have static operands the provide selection of which channel(s), cell(s) or sub-element should be involved when the operation when it is applied. Validation of such operations involves checking to ensure that the selection is only associated with valid channel(s), cell(s) or sub-element. When the selection range is invalid the calculatorElement is invalid, and application of the calculatorElement should not be performed.

The '***in*** ' operation is associated with input channels and selection validation should be performed relative to the number of input channels defined in the header of the calculatorElement.

The '***out*** ' operation is associated with output channels and selection validation should be performed relative to the number of output channels defined in the header of the calculatorElement.

The '***tget'***, '***tput'***, and '***tsav'*** operations are associated with temporary memory cells accessed by and integer index in the range from 0 through 65535. If the operand selection is outside this range the operation is invalid.

The sub-element invocation operations use the operand selection to determine which sub-element to invoke. Therefore, if the selection operand is larger than or equal to the number of sub-elements the operation is invalid. Additionally, sub-element invocation operations that are associated with a specific processing element

type (i.e. '***curv'***, '***mtx*** ', '***clut'***, '***calc'***, '***tint'***) should match the type of the actual indexed sub-element of the calculatorElement or the operation is invalid.

3.2 Branch Validation

The main function of a calculatorElement is made up of an array of operations, and application of the main function starts at the first operation and sequentially applies operations until the array is exhausted. However, a powerful aspect of the calculatorElement is the ability to perform conditional application of operations through branching. (Note: the concept of looping is not defined as part of a calculatorElement main function so the execution of a main function can conceptually be represented as an operation tree).

The data operand of the '***if*** ', '***else'***, '***case'***, and '***dflt'*** branch operators each contain a 32 integer specifying how many operations are part of the branch associated with that operation. An important aspect of main function validation is ensuring that the number of operators defined for each branch are contained within the context that the branch exists in (as branching operations can be nested).

The default context for the main function is all the operators in the main function. When a branch is found the context becomes the operators associated with that branch.

Validation of branches is performed by recursively descending all branches of the operation tree ensuring that the number of operations associated with each branch is less than or equal to the number of operations remaining in its containing context. If any branch contains more operations that are available in its context then the main function is invalid and application of the caclulatorElement should not be performed.

3.3 Operation Data Stack Usage Validation

Nearly all operations in the main function of a calculator element either remove data values from the operation data stack as arguments to the operation or they place the results of the operation as data values onto the data stack.

The number of stack data values involved as either arguments or results is specified for each operation in the iccMAX specification and is static based on either the nature of the operator or the operand data associated with the operation. Thus, stack usage for every operation in the main function can be determined prior to the application of the main function to evaluate whether possible stack underflow or overflow conditions are possible.

Validation of data stack usage is performed by keeping track of the number of values that would be on the data stack for each operation in the operation branching tree (starting with an empty stack at the beginning of the main function). For each operation in the operation branching tree the following two things are determined relative to the stack size:

1. It is determined if application of the operation would have sufficient values on the data stack it needs as arguments (checking for stack underflow)
2. It is determined if the stack size count would exceed the maximum allowable stack size based on the application of the operation (checking for stack overflow).

If either stack underflow or overflow are associated with any operation in the operation branching tree then the main function is invalid and application of the calculatorElement should not be performed.

calculatorElement Application

Validation ensures that calculatorElement application can reliably be performed reliably with minimal risk of invalid operations and undesirable memory access. However, additional runtime requirements are essential to get reliable predictable results.

Application of the main function for an instance of input channels is defined to be singular and predictable. This ensures that the application of calculatorElement for one instance does not impact successive or parallel applications of a calculator element for other instances. This requires that for each application of the main function the data stack is empty, all uninitialized temp channels are set to zero, and any output channels not set by the main function is set to zero.

All operations in the main function are expected to succeed having the exact stack interaction behavior as defined by the iccMAX specification. This means that invalid, infinite, or out of range values are correctly handled by the implementation without causing runtime exceptions.

Caching of temporary memory channel initialization based on CMM environment values is one method to improve performance as environment variables are likely not to change often. When this is done operations are applied up to the point where input channels are accessed, and non-zero values of temporary memory channel values are cached. Application of the main function is then started at the point where the input channels are utilized with temp channels initialized from the cache. The output channel results with caching in place should be identical to the results without caching in place. Additionally, if a CMM supports runtime changing of environment variables then the updating of the cache will need to be updated outside the context of applying the main function on input data.

4 Implementation Evaluation

A suite of iccMAX profiles have been created to help evaluate how well an implementation handles situations which pose security risks relative to the use of a calculatorElement. These profiles can be found in the CalcTest folder associated with the referenceImplementation test files. A discussion of these profiles and expected behavior when trying to apply them follows.

The profiles starting with calcOverMem are profiles that test for temporary memory overruns for specific operators. The profiles starting with calcUnderStack test for stack underflow conditions for various operators. The checkInvalidProfiles.bat file tests the attempted load and application of profiles to determine if the CMM does correct profile validation.